FORCE-DIRECTED SCHEDULING IN AUTOMATIC DATA PATH SYNTHESIS

P.G. PAULIN[1] - J.P. KNIGHT[2]

[1] Dept. 5L40, Bell-Northern Research
P.O.Box 3511, Stn C, Ottawa, ONT. K1Y 4H7
[2] Carleton University, Colonel By Dr, Ottawa, ONT. K1S 5B6

ABSTRACT

The HAL system performs data path synthesis using a new scheduling algorithm that is part of an interdependent scheduling and allocation scheme. This scheme uses an estimate of the hardware allocation to guide and optimize the scheduling subtask. The allocation information includes the number, type, speed and cost of hardware modules as well as the associated multiplexer and interconnect costs.

The iterative **force-directed** scheduling algorithm attempts to balance the distribution of operations that make use of the same hardware resources:

- **Every** feasible control step assignment is evaluated at each iteration, for **all** operations.
- The associated side-effects on **all** the predecessor and successor operations are taken into account.
- All the decisions are global.
- The algorithm has $O(n^2)$ complexity.

We review and compare existing scheduling techniques. Moderate and difficult examples are used to illustrate the effectiveness of the approach.

1. INTRODUCTION

In the automatic design of application specific integrated circuits, the algorithmic description is commonly synthesized into a datapath and a control path. Scheduling datapath operations into the best control steps is a task whose importance has been recognized in many systems [1,2,3,4,5]. According to Gajski [1], it is "perhaps the single most important step during the architecture synthesis". Ironically, it is also the one that has received the least attention in current literature. (Notable exceptions are the papers presented by Alice Parker's group at the University of Southern California [2,3] and by Emil Girczyc of the University of Alberta [4]).

Operation scheduling determines the serial/parallel nature of the design and approximates cost-speed trade-offs [6]. If the design is subjected to a speed constraint, the scheduling algorithm will attempt to make sufficient operations run in parallel to meet the constraint. Conversely, if there is a limit on chip area, the scheduler can be asked to serialize operations to give the maximum speed consistent with the constraint.

The scheduling task is an important one as it affects four fundamental aspects of the subsequent synthesis process:

- The number and type of processors allocated
- The timing constraints on these processors
- The storage requirements
- The data transfer requirements

The HAL system is composed of three main modules that were described in [5]. The InHAL module uses a novel **force-directed** scheduling algorithm that attempts to balance the distribution of operations that make use of the same hardware resources. This algorithm also allows specific allocation information, such as the propagation delay of a specific standard cell, to be used when optimizing the scheduling process. The MidHAL module performs the hardware allocation while the final interconnection and optimization step is realized by the ExHAL module.

We will first review existing scheduling techniques and compare them with the approach used in the HAL system. This will be followed by a description of the force-directed scheduling algorithm that is the main emphasis of this paper. We will then demonstrate how processor allocation information is integrated into the scheduling process. Finally, we present experimental results for moderate and difficult problems taken from current literature.

2. LITERATURE SURVEY

The simplest way to perform scheduling is to relegate the task to the user. This is the approach favored by the Silc system [7] under the assumption that the user should explicitly define the parallelism of the design.

Independent scheduling/allocation schemes:

The next simplest scheme is to schedule operations "as soon as possible" (ASAP) as is done in the Emerald/Facet system [8]. This technique has proved useful in the past for near-optimal microcode compaction [9].

A refinement of this concept is ASAP scheduling with conditional postponement of operations. In the MIMOLA system [10], this occurs whenever the operation concurrency is higher than the number of available processors. The recently published Flamel system [17] uses the same idea. The scheme used in Kung, Whitehouse and Kailath's book on digital signal processing [11] is similar, except that an operation is postponed when it blocks a later one with a lower "as late as possible" (ALAP) level.

Continuing along the scale of increasing complexity, we have the algorithms that use **list scheduling**, as described in [9]. The behavioral synthesis of interfaces (BSI) system developed by Nestor [12], the SLICER system developped by Pangrle and Gajski [13] and the BUD-DAA system [14] use this type of approach.

In list scheduling, operations are sorted in topological order (top to bottom) by using the precedence relations dictated by data and control dependencies. The sorted operations are then iteratively scheduled into control steps. The order in which they are placed into a control step is determined by a heuristic priority function that is applied to all operations that can be placed in the control step.

In the BSI system, the priority function reflects whether placing the operation in the current step will violate a minimum time constraint (ASAP time) and whether placing an operation in a later step will violate a maximum time constraint (ALAP time).

[1] This research was funded in part by grants from the Natural Sciences and Engineering Research Council, Canada (NSERCC) and from Bell-Northern Research, Ottawa.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0738-100X/87/0600-0195$00.75

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

In the SLICER system, the priority function is based on operation **mobilities**. The mobility of an operation is defined as the difference between it's ASAP time and ALAP time. Operations on the critical path are thus scheduled first.

The BUD-DAA system uses a similar priority function but sorts the operations in depth-first order (bottom to top).

Strictly speaking, the scheduling is not totally independent of the allocation in these systems. Most derive estimates of the operation delays based on the most likely processor assignments.

Interdependent scheduling/allocation schemes:

In these systems, the operation scheduling is done concurrently with the processor allocation (except for the Silc system, where the user has final say). Elf uses a variation of the list scheduling algorithm where the priority function is based on operation **weights** and **urgencies**. Operation weights are calculated by taking the minimum number of cycles to execute the operation plus the maximum weight of its successors. Operation urgencies are determined by taking a ratio of the operation's weight and the number of cycles left until its time constraint. When an operation is delayed its urgency increases which raises it's priority for assignment in the next control step.

The elaborate scheme used in USC's MAHA system [2] relies on critical path determination and the concept of operation **freedom** to guide scheduling. The freedom of an operation is identical to the mobility calculated in the SLICER system described earlier.

The MAHA system first invokes the Clocking Scheme Synthesis Package (CSSP) written by Park [3]. Here the critical path is determined and divided optimally into n steps, one per clock cycle. The clock cycle is also automatically determined here. In the HAL system, the clock cycle must be defined by the user.

MAHA then allocates processors for the critical path in a first-come first-served fashion. The notion of freedom is used to guide the scheduling of the off-critical-path nodes. The node with the smallest freedom is chosen for allocation. The critical path assignment is completed in linear time. The assignment of off-critical-path nodes is done in $O(n^4)$ time in the worst case.

We can thus summarize the approaches:

- Independent scheduling/allocation
  - ASAP scheduling approaches:
    - — Direct (Facet-Emarald)
    - — Conditional deferment (MIMOLA, Flamel, [11])
  - List scheduling approaches:
    - — Priority function: time constraints (BSI)
    - — Priority function: mobility (SLICER)
    - — Priority function: critical path, use of depth-first ordering (BUD-DAA)
- Interdependent scheduling/allocation
  - User-defined schedule (Silc)
  - List scheduling using urgency (ELF)
  - Freedom and critical path scheduling (MAHA)
  - Force-directed scheduling (HAL)

3. SYSTEM COMPARISON

The HAL system's approach to scheduling is fundamentally different from all of the approaches discussed above. We describe here three of the most important differences.

Local vs global evaluations:

This is, in our view, the most important novel contribution of the HAL system to the scheduling/allocation task. Virtually all the systems discussed rely on a local evaluation of the effect of the control step assignment of the current operation considered.

The HAL system considers the related effects of a control step assignment on all the predecessor and successor operations. Moreover, the assignment's impact on the **overall** concurrency of operations is evaluated through the use of distribution graphs (DG). These DGs represent the current concurrency of each class of operation at each control step (c-step from now on). Unscheduled operations also contribute to the determination of the operation concurrency except that their contribution is probabilistic rather than deterministic.

Finally, **every** feasible assignment of each operation to any of its possible c-steps is considered at each iteration.

While this global approach would seem to imply large computation costs, the algorithm is $O(n^2)$, where n is the number of operations. This relatively low order of complexity is due to an efficient method of propagating predecessor and successor forces. We shall give a brief description of this method in section "4.3 ORDER OF COMPLEXITY".

Independent vs interdependent scheduling/allocation:

In the Facet, MIMOLA, BSI, SLICER and BUD-DAA systems the scheduling is performed more or less independently of the processor allocation. The Elf and MAHA systems do both tasks concurrently.

The HAL system, on the other hand, does both tasks separately but not independently. The operation scheduling and processor allocation are determined by stepwise refinement as depicted in Figure 8 of section "5. SCHEDULING/ALLOCATION BY STEPWISE REFINEMENT".

Topological vs middle-out scheduling:

The final difference relates to the order of selection of operations during the scheduling process. In all the systems mentioned above (with the exception of MAHA) , the operations are scheduled iteratively by using their topological order (depth-first order in BUD). The MAHA system first schedules the operations on the critical path and then schedules the remaining ones in order of increasing **freedom**.

In the HAL system, the ordering is based on the lowest **force**. As we shall see in the next section, this value is independent of the precedence ordering. It may be considered a 'middle-out' approach.

4. FORCE-DIRECTED SCHEDULING ALGORITHM

The selection of the operation to be scheduled, and of the c-step to which it will be assigned, is based on an evaluation of the move which causes the most balanced distribution of operations in each c-step. This algorithm is an iterative one in which one operation is scheduled at each iteration. The intent is to reduce the number of processors required by reducing the concurrency of the operations assigned to them, but without lengthening the total execution time.

4.1 BASIC SCHEDULING ALGORITHM

Determination of time frames:

The first step involves the determination of both an ASAP (as soon as possible) scheduling and an ALAP (as late as possible) scheduling. Combining results for both schedules will determine the possible time frames for each operation. To illustrate this, we will use the example given in [5]. The DFG derived from this example is given in Figure 1. The raw DFG would not have operations scheduled in time (control steps). Here, ASAP scheduling is shown. For simplicity, it will be temporarily assumed that all operations require one clock cycle and that succeeding operations cannot be scheduled in the same cycle.

The ASAP and ALAP operation scheduling are shown (in simplified form) in Figure 2.

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

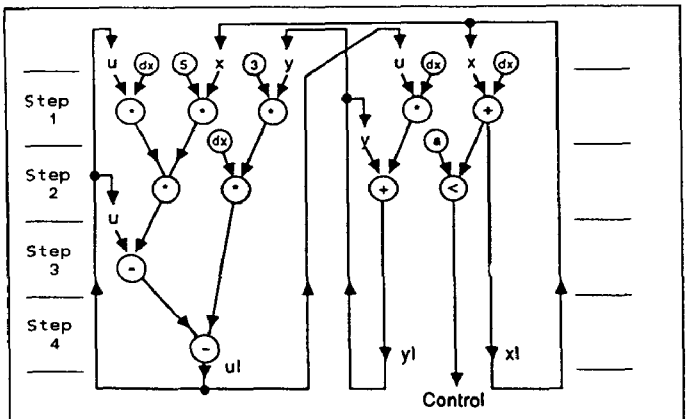

Figure 1. Data flow graph for first example

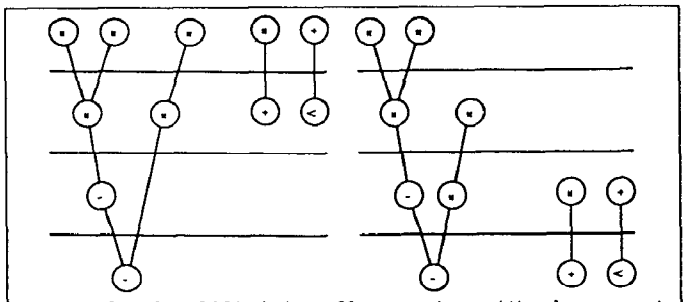
Figure 2. Simplified data flow graphs with loops not shown. They are used to illustrate ASAP and ALAP scheduling.

By noting that the actual scheduling of a particular operation may be anywhere between it's ASAP cycle and it's ALAP cycle, one can draw a time frame diagram (Figure 3). Here the width of the box containing a particular operation represents the probability that the operation will be eventually placed in a given time slot. A useful heuristic is to assume uniform probability of placing an operation in any feasible control step. The area of each operation is always one, but it is 'stretched' along its time frame.

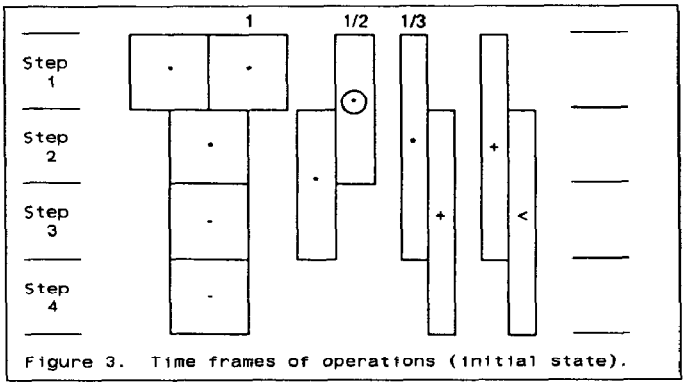

Figure 3. Time frames of operations (initial state).

Creation of distribution graphs (DGs):

The next step is to take the summation of the probabilities of each type of operation for each control step of the DFG. The resulting distribution graphs (one for each type of operation) indicates where concurrency of similar operations is high and will direct the scheduling algorithm accordingly.

Conditional statements:

The presence of conditional statements (e.g. if-then-else and case statements) causes some operations to be mutually exclusive. When these operations can be executed on the same processor type (i.e. assigned to the same DG), they can be scheduled into the same c-step without increasing the processor allocation. The same processor is simply shared by those operations as they will never be executed concurrently.

We use the following stratagem to take advantage of this observation: for each c-step in which the time frames of the mutually exclusive operations intersect, the probability of only one of these is added to the corresponding DG. The operation selected is the one with the highest probability. This is illustrated for the simple DFG of Figure 4. The fork and join operations correspond to an if-then-else statement.

Without special treatment of the mutually exclusive additions, the total distribution would be 1.5 in both c-steps. The unscheduled addition would then have an equal probability of being assigned to either c-step. It is obviously preferable to schedule it in the first c-step, as in this case only one adder will be required. This is exactly what will happen using the stratagem just described due to the reduced distribution in the first c-step.

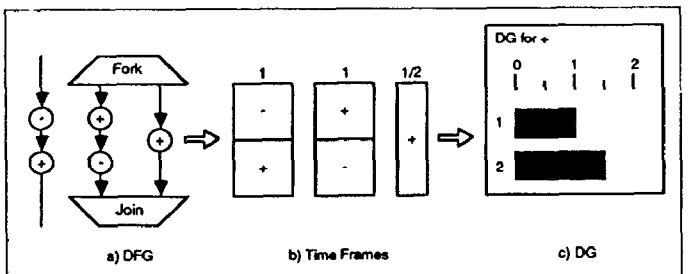

Figure 4. Distribution evaluation for DFG with conditional statement.

The distribution graphs derived from Figure 3 are given below in Figure 5.

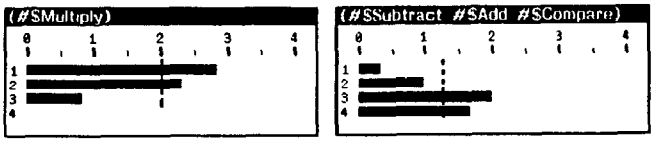

Figure 5. Distribution graphs (initial state).

The first graph represents the distribution of the multiply operations, and the next one combines the distributions of the add, subtract and compare operations. The latter three operations are actually assigned to separate DGs but are grouped here for conciseness. Each horizontal bar of the DGs corresponds to a distinct c-step.

Calculation of 'self' forces:

Each operation of the DFG will have a **force** associated with each cycle of it's time frame. This is a quantity which reflects the effect of an attempted control step assignment on the overall operation concurrency. It is positive if the assignment causes an increase of operation concurrency and negative for a decrease.

The force is much like that exerted by a series of 'springs' that obey Hooke's law; **F = Kx**. **K** represents the spring's constant (rigidity), **x** the displacement and **F** the force caused by the displacement.

Each distribution graph will have associated springs (one for each c-step) that will exert forces on all operations. The constant of the spring **K** is represented by the value of DG(i) where i is the c-step number for which the force is calculated. The displacement of the spring **x** is given by the increase (or decrease) of the probability of the operation in the c-step due to a rescheduling of the operation.

We will illustrate this by using the partial time frame diagram of Figure 6. The constant of each of the three springs corresponds to the value of the multiplication DG given in Figure 5 for the first three c-steps.

We will attempt to schedule the circled multiply operation in c-step 1 as depicted in Figure 6 b). The probability of the operation will change from 1/2 to 1 in c-step 1 and from

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

1/2 to 0 in c-step 2. These probability shifts correspond to the displacement x of each of the 'springs'. The resulting force associated with the move to c-step 1 is the sum of forces in both c-steps and is given by:

$$\begin{aligned} Force(1) &= Kx(1) + Kx(2) \\ &= DG(1) * x(1) \quad + \quad DG(2) * x(2) \end{aligned}$$

Using the values from Figure 5 and Figure 6 we obtain:

$$\begin{aligned} Force(1) &= (2.8333*0.5) - (2.3333*0.5) \\ &= +\ 0.25 \end{aligned}$$

The force is positive as expected because the concurrency in c-step 1 is higher than in c-step 2. Scheduling the multiplication in that c-step will have an adverse effect on the overall distribution.

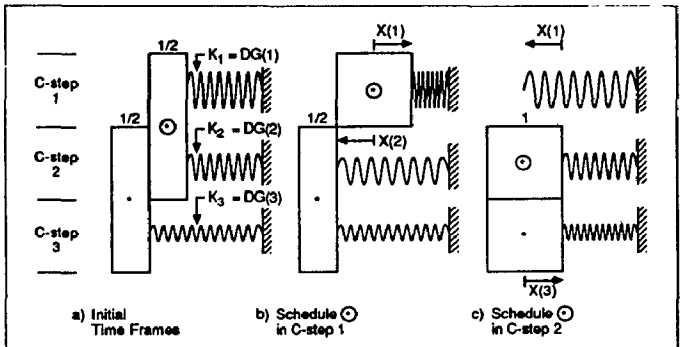

Figure 6. Time frame modifications for force calculations.

Calculation of predecessor and successor forces:

Assigning an operation to a specific c-step will often affect the time frames of linked operations in the DFG. In turn, this will create additional forces that can reduce or even counter the original intended improvement so it is imperative that they be accounted for.

This is achieved by calculating the extra forces due to the implicit rescheduling of linked operations. They shall then be added to the 'self' force. There will be two extra force contributions: the predecessor forces and the successor forces.

For example, if the circled multiply operation in Figure 6 a) was tentatively assigned to c-step two, as in Figure 6 c), the succeeding multiply operation would implicitly be assigned to c-step three. The time frames of the other operations wouldn't be affected. The resulting force associated with that assignment would then be the sum of the two individual forces. The force of the first multiply is given by:

$$\begin{aligned} Force'(2) &= (DG(1) * x(1)) + (DG(2) * x(2)) \\ &= -(2.8333*0.5) + (2.3333*0.5) \\ &= -\ 0.25 \end{aligned}$$

The successor force of the second multiplication is given by:

$$\begin{aligned} Force"(2) &= (DG(2) * x(2)) + (DG(3) * x(3)) \\ &= -(2.3333*0.5) + (0.8333*0.5) \\ &= -\ 0.75 \end{aligned}$$

And the resulting force of the c-step assignment is:

$$\begin{aligned} Force(2) &= Force'(2) + Force"(2) \\ &= -1 \end{aligned}$$

This c-step assignment is much more advantageous than the one attempted earlier, and this is reflected clearly by the force calculations.

Selection of best move:

The self, predecessor and successor forces are calculated for each possible c-step of all the operations. The operation and c-step pair selected is the one with the most negative force (or the lowest positive force). The operation's time frame is then reduced to the selected c-step and the linked operations' time frames are modified accordingly.

We can thus summarize the entire process:

1. Determination of time frames
2. Update of DGs (with conditionals taken into account)
3. Calculation of 'self' forces
4. Calculation of predecessor and/or successor forces
5. Scheduling of best operation and c-step pair

These steps are repeated until the time frame of each operation is reduced to one. The distribution graphs are updated at each iteration to reflect the current operation distributions. The forces on the remaining unscheduled operations will vary accordingly.

Sample scheduling:

Figure 7 depicts the time frames and the final distributions for the example of Figure 1.

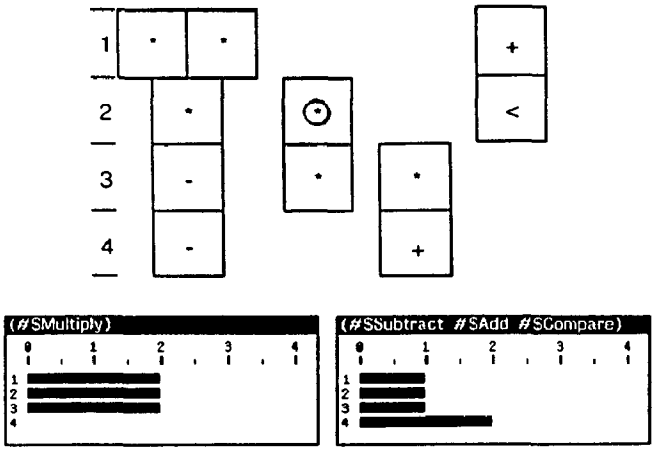

Figure 7. Final time frames and DGs

4.2 REFINED SCHEDULING ALGORITHM

Multiple operation DGs:

In many cases we have to deal with multi-function processors (e.g. Alus). To deal with this, the concept of distribution graphs (DG) was extended so that the distribution of one or more operation types can be stored in a single DG.

For example, if the allocator assigns additions and subtractions to an Alu, then a two class DG will be created in lieu of the two single class ones. These two types of operations will then tend to be scheduled in different clock cycles to make best use of the Alu.

Multiple operations per cycle and multi-cycle operations:

The possibility of scheduling multiple dependent operations in a single cycle has also been incorporated to the system. This feature is implemented in a straightforward fashion by extending the time frames of fast combinatorial operations *into the previous and/or next cycles (when the total delay* in those cycles is less than the cycle time). Multi-cycle operations are also supported and are implemented with a simple extension of the single-cycle methodology.

Incorporation of processor costs:

Different processor cells have different realization costs. Operation types associated with high-cost processor cells should be given higher priority in the scheduling process. An easy way to do this is to multiply the constant of the 'spring' associated with the DG by a **cost factor** that is a function of the processor area and the associated mux and interconnect costs. This function will be given in section "5.3 USE OF ALLOCATION INFORMATION FOR SCHEDULING".

Note on storage requirements:

The minimum number of registers required for the implementation of a given DFG is given by the largest number of data arcs traversing a control step boundary.

Most of these data arcs are initiated by operation nodes. As the force-directed scheduling algorithm attempts to distribute these operations evenly across the DFG, the number of arcs traversing a given control step is also balanced. In turn, this reduces the minimum number of registers required.

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

For the ASAP scheduling shown in Figure 1 at least seven registers are required (there are seven values to be stored at the end of cycle one). For the optimized scheduling shown below, this number is reduced to five (although in the final synthesized data path, six registers were used to save on interconnect).

Although these observations are encouraging, a more explicit treatment of register allocation will still be required in the future. For more information on the HAL system's present approach to register allocation, refer to [5].

4.3 ORDER OF COMPLEXITY

It can easily be seen that the algorithm described above is $O(n^3)$ when implemented in a straightforward fashion (n is the number of operations). Fortunately, we have found an alternate method of including the forces of predecessor/successor operations that can be applied without any loss of generality. In this method, the self, predecessor and successor forces are calculated in three separate passes.

In the first pass the force applied directly to each operation (its self force) is calculated for every c-step in its time frame. This calculation is done in linear time. These values are then stored in a vector associated with the operation.

In the second pass, the DFG is traversed in depth-first order. The total force for each operation is given by the sum of its self force and the stored total force of its direct successors only. In this way we will effectively be performing a running sum of all the successor forces in linear time. In the third pass the process is repeated for the predecessor forces by taking the operations in topological order.

Using this method, the order of complexity of the algorithm is now $O(n^2)$. This is accomplished at the expense of a slight increase in computer memory utilization.

5. SCHEDULING/ALLOCATION BY STEPWISE REFINEMENT

The scheduling/allocation iterative loop used in the HAL system attempts to reconcile two conflicting goals:

- The optimal scheduling of operations without explicit foreknowledge of the processor allocation.
- The optimal allocation of processors without exact information on the concurrency of operations or their propagation delays.

This is accomplished by stepwise refinement of the operation scheduling and processor allocation as depicted in Figure 8.

5.1 SCHEDULING/ALLOCATION PHASES

Phase 1 Default Allocation: Allocation of default single-function processors to perform each type of operation. The fastest processors are temporarily assumed.

Phase 2 Preliminary Schedule: Balancing of the distribution of similar operations. Default processor speed and area costs are used.

Phase 3 Refined Allocation: Allocation of single and multi-function processors with relaxed constraints on processor speed.

This can be done as it now has a better estimate of:

- The timing constraints on the individual operations (they may have been relaxed on some operations).
- The overall concurrency of operations of different classes.

As depicted in Figure 8, the mux and interconnect area costs are also evaluated here.

Phase 4 Final Schedule: Balancing of the distribution of operations requiring similar processor types. The number, type, speed and area of the allocated processors are used here to guide the scheduling.

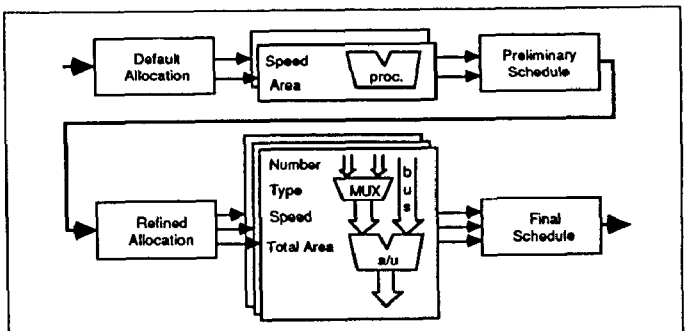

Figure 8. Scheduling/Allocation by stepwise refinement

Before going on to describe how scheduling and allocation are linked, we will give a short description of the module (named MidHAL) that performs the allocation task. A more elaborate one is given in [5].

5.2 THE MIDHAL ALLOCATER

This module is implemented as a rule-based expert system. It accepts as input the DFG, a global time constraint, the local time constraints on each operation and a library of available processors and their associated speed and cost. It then uses heuristic rules to select a set of locally optimum processors to execute the operations described in the DFG, given the time constraints on these operations.

The following information is returned:

- The type of processors allocated
- The number of processors of each type
- Their propagation delay
- Their area cost (including mux and interconnect area)

The mux and interconnect area cost is evaluated by partitioning the DFG and doing a preliminary assignment of operations to processors. The data transfer requirements can then be evaluated. The minimum number of multiplexers and interconnections required to perform these data transfers is then estimated.

The function and intent of this module is quite similar to that of the BUD module in the recent BUD-DAA system [15]. However, as preliminary floorplanning is not performed in the HAL system, the interconnect area estimation will not be as precise.

Moreover, the HAL system allows the allocation task to be performed completely or partially by the user. The system will verify that the allocated hardware is sufficient and will add processors as necessary. In this way the user can invoke the scheduling/allocation loop repeatedly with different processor selections and explore the design-space semi-automatically.

5.3 USE OF ALLOCATION INFORMATION FOR SCHEDULING

We shall show here how the four types of information returned by the MidHAL module can be used to guide the scheduling process.

Types of processors allocated:

For each type of procesor allocated, one single distribution graph (DG) is created. Multi-function processors such as Alus will be assigned to the multiple operation DGs described in section "4.2 REFINED SCHEDULING ALGORITHM"

Number of processors allocated:

In most cases, the MidHAL module allocates the minimum number of processors for every type of operation. This number is equal to the maximum number of concurrent operations they perform. But there are also cases where a larger number of processors are allocated. This may happen when the cost saving due to processor sharing is offset by associated multiplexer and interconnection costs. We must include this information so that the forces applied by the assigned DG are reduced, as more than enough processors are likely to be available.

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

More specifically, for each processor type (i.e. for each DG) and each c-step we calculate the unused processor capacity. This is given as the difference between the number of allocated processors and the current number of operations scheduled in c-step i.

$$\text{Unused Capacity}(i) = \text{No Processors} - \text{No Opns}(i) \quad (1)$$

The new biased DG value (i.e. the new spring constant) is then given by the old value less the unused processor capacity.

$$\text{Biased DG}(i) = \text{DG}(i) - \text{Unused Capacity}(i) \quad (2)$$

When the number of scheduled operations in c-step i reaches the number of processors allocated, DG(i) will take on its original value.

Processor speed:

This information is integrated in a straightforward fashion by assigning the propagation delay of each processor to the operations it performs. The determination of time frames and subsequent scheduling will thus be directly affected by these new values.

Processor, multiplexer and interconnect costs:

As we mentioned in section "4.2 REFINED SCHEDULING ALGORITHM" each DG element is multiplied by a cost factor. For the initial scheduling in phase 2, this cost factor is simply the processor area:

$$\text{Cost Factor} = \text{Processor Area} \quad (3)$$

For the final scheduling in phase 4, we can use the estimate of the mux and interconnect area derived in phase 3 to calculated a weighed cost factor:

$$\text{Cost Factor} = \text{Processor Area} * \text{Interconnect Factor} \quad (4)$$

The interconnect factor represents the relative importance of the processor area with respect to the combined processor, mux and interconnect area:

$$\text{Interconnect Factor} = \frac{\text{Processor Area}}{\text{Total Area}} \quad \text{where} \quad (5)$$

$$\text{Total Area} = (\text{Processor} + \text{Mux} + \text{Interconnect}) \text{ Area} \quad (6)$$

For high-cost processors such as multipliers the interconnect factor tends towards one. Alternately, for simple processors (e.g. logic gates) the mux and interconnect area can be much larger than the processor area in which case the interconnect factor tends towards zero.

In general, this will result in an increased number of low cost processors, but the associated area increase will be offset by reduced multiplexer and interconnect costs. On the other hand, sharing of high-cost processors will still be encouraged.

5.4 SCHEDULING UNDER FIXED HARDWARE CONSTRAINTS

The scheduling and allocation approaches presented above support the synthesis of near-minimal cost datapaths under fixed timing constraints. A simple extension of this methodology also allows the determination of a schedule with a near-minimal number of c-steps given fixed hardware constraints.

The basic idea consists of performing regular force-directed scheduling with a tight timing constraint. During the scheduling, hardware resource costs are estimated at every iteration. Whenever a hardware constraint is exceeded (this can be a constraint on the number of processors of any type, or a constraint on the total hardware cost), then an extra c-step is added to the graph. Thus the time frames of scheduled and unscheduled operations are extended by one c-step.

This increases the time required to arrive at a solution, but not the order of complexity of the algorithm. This is because the partial scheduling is maintained, i.e. the time frames of operations are extended, not recalculated.

This extra capability will be illustrated in the example of section "6.3 TEMPERATURE CONTROLLER FROM ELF".

5.5 SAMPLE SCHEDULING AND ALLOCATIONS

The processor allocations for the example given in section "4. FORCE-DIRECTED SCHEDULING ALGORITHM" are tabulated in Figure 9 for different time constraints. The refinements of the scheduling algorithm described above were applied here. (Note: The multiplications are given a more realistic propagation delay of two cycles. The critical path is now six cycles long.) The CPU execution times given are for a XEROX 1108 Lisp machine.

COMPONENT	(* cost > alu cost)				(alu cost > * cost)			
No Cycles	6	7	8-12	13	6'	7'	8'-12'	13'
multiplier (*)	(*) (*) (*)	(*) (*)	(*) (*)	(*)	(*) (*) (*) (*)	(*) (*) (*)	(*) (*)	(*)
alu (a)	(a) (a)	(a) (a)	(a)	(a)	(a)	(a)	(a)	(a)
CPU (sec)	15	35	65-120	180	15	30	65-125	175

Figure 9. Processor allocation for all time constraints: Example of Figure 1.

The results in the first four columns illustrate the realistic case where the cost of the multiplier is higher than the cost of the Alu. The results in the next four columns represent an illustrative example where the cost factor of the Alu is set artificially higher than the cost of the multiplier. The change in forces resulted in the decrease of the concurrency of the Alu operations at the expense of the multiplication operations. This is exactly the trade-off that we are hoping for.

The results obtained in column 6''and 7' are identical to the ones that would be obtained if the MidHAL module (the processor allocater) had allocated four and three multipliers respectively. This is due to the associated reduction of the multiplication forces dictated by equation (2) of section "5.3 USE OF ALLOCATION INFORMATION FOR SCHEDULING", such that the concurrency of the Alu operations is reduced to one. Once again, this trade-off is the one that should follow logically.

6. EXPERIMENTAL RESULTS/COMPARISONS

The four examples presented in this section are taken from some of the systems described in section "2. LITERATURE SURVEY". They where chosen to illustrate the flexibility of the HAL system with respect to considerably different constraints and also to allow comparison with the results obtained from these systems.

Experimental Procedure:

For each of the examples presented, the scheduling was performed using the same assumptions as the original reference. They are listed at the beginning of each subsection. The results were obtained without any fine tuning of the algorithm to the examples. The CPU execution times given are for a XEROX 1108 Lisp machine. This is a single-user workstation in the medium-low performance range (by today's standards).

6.1 EXAMPLE FROM MAHA

This example was presented in [2] and makes uses of the CSSP program [3] for the initial stage partitioning. Two results are given in the paper, the fastest allocation and the cheapest one. For the fastest allocation, the graph is partitioned into four stages (clock cycles). For the cheapest allocation, the graph is partitioned into eight stages.

Assumptions:

- All similar operations have equal propagation delays.
- For the fastest allocation: the clock cycle is such that a maximum of three operations can be combined in a single cycle.
- For the cheapest allocation: only one operation can be performed per cycle.

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

- There is no sharing of processors executing mutually exclusive operations (i.e. operations that are on either side of an if-then-else statement).
- MAHA execution times are for a VAX 11/750.

The table below gives the allocation for different timing constraints. These constraints are the number of cycles (stages) and the maximum number of operations per cycle. The results given in the last of the three sets of columns are for the HAL system when mutually exclbsive operations (due to conditionals) are taken into account. This is using the method described earlier in section "4.1 BASIC SCHEDULING ALGORITHM".

COMPONENT	MAHA		HAL			HAL (w. mutual exclusion)		
Cycles	8	4	8	4	3	8	4	3
Opns/Cycle	1	3	1	2	3	1	2	3
subtractor (-)	(-)	(-) (-) (-)	(-)	(-) (-)	(-) (-) (-)	(-)	(-)	(-) (-)
adder (+)	(+)	(+) (+)	(+)	(+) (+)	(+) (+) (+)	(+)	(+) (+)	(+) (+)
CPU (sec)	160	160	50	25	35	85	55	75

Figure 10. Example from MAHA[2].

For the cheapest allocation (8 cycles, 1 Opn/cycle), the HAL system arrives at the same number of processors as the MAHA system. This was expected (hoped for) as all processors have 100% utilization.

For MAHA's fastest allocation (4 cycles, 3 Opns/cycle), the HAL system allocates one subtractor less, but with a tighter constraint on the clock cycle (2 opns/cycle). In this case the reduction in area (with respect to MAHA) is close to 20% with a 25-30% reduction of the clock cycle time (the actual value will depend on the latch propagation delays). Processor utilizations are 100%.

The fastest realization in the HAL system runs in three cycles (3 opns/cycle) and uses three adders and three subtractors. This solution requires nearly 20% more area than MAHA's fastest realization but the number of required cycles is reduced by 33%.

Taking mutual exclusion into account yields the most impressive results by far (given in the third set of columns). This example certainly illustrates the potential importance of doing so.

6.2 PIPELINED FIR FILTER FROM SEHWA

The results presented here are for the pipelined 16-point digital FIR filter example borrowed from [15]. The SEHWA system described in this paper does scheduling and allocation of pipelined data paths.

Pipeline scheduling is achieved in the HAL system by folding the DGs across the boundaries determined by the latency value (in cycles). This will cause the force-directed scheduling algorithm to balance the distribution of concurrent pipeline stages. For lack of space, we must defer the detailed explanations to a forthcoming paper [16].

Assumptions:

- The maximum stage time limit is 100 ns.
- The latency is equal to 300 ns.
- Additions are performed by adders with a delay time of 40 ns.
- Multiplications are performed by multipliers in 80 nsec.
- The latch delays are 20 ns.
- Multiple succeeding operations in a single cycle are allowed.

In Figure 11 we present the results for three scheduling methods:

1. SEHWA: Backward feasible scheduling.
2. SEHWA: Exhaustive feasible scheduling.
3. HAL: Force-directed pipeline scheduling.

These include the order of complexity of the algorithm, the number of cycles required, the number and type of processors allocated, and the CPU time. Only an approximate run time (on a VAX/750) was given in [15].

SYSTEM	SEHWA		HAL
Algorithm	Feasible	Exhaustive	Force-Directed
Complexity	$O(n^2 \log n)$	$O(n^{m_r})$	$O(n^2)$
Number of Cycles	7	6	6
Number of Adders	6	5	5
Number of Multipliers	3	3	3
CPU time	n/a	< 1 hour	30 sec

Figure 11. FIR filter results for SEHWA Example

We see that the force-directed pipeline scheduling technique yields a result similar to that obtained by the exhaustive scheduling (i.e. an optimal result).

Futhermore, the result obtained with the backward feasible scheduling is more expensive (six vs five adders) and slower (seven vs six clock cycles) although it's complexity is actually higher than that of the force-directed algorithm.

6.3 TEMPERATURE CONTROLLER FROM ELF

In this paper, the system was used to find the minimum number of clock cycles required given specified timing and hardware constraints. Two examples where given.

Assumptions for the first example:

- The operations of the DFG where divided into two groups with two different timing constraints.
- An initial processor library is given. It contains adders, subtractors, a comparator, a shifter (for division) and an Alu.
- The solution must not require more than one processor of each type.
- All operations (except divisions) have a one cycle propagation delay.
- Division operations require a two cycle delay.
- Succeeding (data dependent) operations cannot be performed in the same clock cycle.
- The inputs are made available sequentially (i.e. one input is made available in each of the first five cycles).

The Elf system generated a solution that required a total of 20 clock cycles, 12 cycles for the first group of operations and 8 for the second. In this version of the Elf system, operations in the second group cannot overlap with operations in the first. The current (unpublished) version does not have this constraint.

Given the same constraints, the HAL system arrived at a solution that required a total of 19 cycles (11 clock cycles for the first group and 8 for the second). Moreover, if operations in both groups are allowed to overlap, but without violating the separate constraints, the system arrives at a solution requiring a total of 15 cycles. It can be shown that these results are optimal, given the assumptions listed above.

Assumptions for the second example:

The assumptions here are the same as above except that a user-defined Alu (performing add, subtract and shift operations) is to be used. Once again, the solution must not require more than one processor of each type.

The number of cycles required by the Elf system was not specified. The HAL system scheduled the operations of the first and second groups into 13 and 8 cycles respectively. These are also optimal results. Moreover, with overlap, the total time required is still 15 cycles, with all constraints respected. These results are mostly due to the optimal use of the Alu through the use of the multiple operation DGs described in "4.2 REFINED SCHEDULING ALGORITHM".

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.

The table of Figure 12 below gives the number of cycles required by both systems for the examples described. The second total number of cycles given (in the HAL column only) corresponds to the case where operations of both groups are allowed to overlap.

Min no of cycles	No Initialisation		Alu Specified	
	Elf	HAL	Elf	HAL
Group 1	12	11	n/a	13
Group 2	8	8	n/a	8
Total	20	19/15	n/a	21/15
CPU(sec)	n/a	100	n/a	80

Figure 12. Minimum number of clock cycles required fordifferent hardware constraints : Elf temperature controller example.

6.4 FIFTH-ORDER DIGITAL 'WAVE' FILTER

The DFG used in this example is borrowed from [11] and implements a fifth-order wave digital elliptic filter. This is a much more substantial example that contains 43 operations (additions and multiplications) submitted to over 60 precedence constraints. It is assumed that additions require five clock cycles and multiplications ten. They were respectively assigned one and two c-steps. The critical path is thus 17 c-steps long (85 cycles).

The table shown in Figure 13 summarizes the allocation of processors for the example. In the last row the CPU execution time is given in minutes (again for a XEROX 1108 Lisp machine). The 17 c-step result (3 multipliers and 3 adders) is to be compared with that given in [11] (4 multipliers and 4 adders). Moreover, the flexibility of the system allows to explore alternate speed-cost trade-offs.

The 21 c-step result (1 multiplier and 2 adders) is a good example of a substantial area saving (over 50%) against a small loss in speed (less than 20%) as compared to the 17 c-step result. This area saving is even greater when compared to the 17 c-step result given in [11].

COMPONENT	[11]	HAL				
No Cycles	17	17	18	19	20	21
adder (+)	(+) (+) (+) (+)	(+) (+) (+)	(+) (+) (+)	(+) (+)	(+) (+)	(+) (+)
multiplier (*)	(*) (*) (*) (*)	(*) (*) (*)	(*) (*)	(*) (*)	(*) (*)	(*)
CPU (min)	n/a	1	3	7	10	13

Figure 13. Processor allocation for all time constraints: Digital wave filter example.

This example is eloquent proof that the system can solve relatively difficult scheduling problems in a reasonable time. A close examination of processor utilizations shows that all but one of the solutions found are guaranteed optimal. Exhaustive scheduling would be needed to determine if this is also the case for the remaining one.

Our experience is that it is nearly impossible to solve this problem manually and obtain results of the same quality. This is mostly due to the large number of interactions between operations (i.e. 43 operations submitted to over 60 precedence constraints). This problem is compounded by the fact that the number of possible schedules grows exponentially with the number of cycles allowed.

On the other hand, the processing time grows linearly with the number of cycles; the complexity of the algorithm is $O(c)$ for a given n, where c is the average number of cycles of all time frames.

7. CONCLUSION

The methodology presented features a novel force-directed scheduling algorithm that is invoked in a four phase scheduling/allocation scheme. We have shown how this approach makes it feasible to incorporate explicit allocation information to optimize the scheduling process. The information transfer is made possible by stepwise refinement of the scheduling and allocation tasks.

The force-directed scheduling algorithm at the heart of this process was shown to run in n^2 time. In spite of this relatively low complexity, the algorithm explores the search space in a global fashion and produced optimal results for nearly all of the examples presented.

Results obtained where shown to be as good or better as the ones previously published. Furthermore, the flexibility of the system was highlighted by the variety of constraints and requirements it had to deal with. These include:

- Multi-cycle operations.
- Multiple operations per cycle.
- Mutually exclusive operations.
- Pipelined data flow graphs.
- Optimized use of single and multi-function processors (Alus).
- Optimization of required hardware for a specified time constraint.
- Optimization of system speed for specified hardware resources.

8. REFERENCES

1. D.D. Gajski, N.D. Dutt and B.M. Pangrle, "Silicon Compilation (Tutorial)", Proceedings of the IEEE 1986 Custom Integrated Circuits Conference, Rochester NY, May 1986, pp. 102-110.
2. A.C. Parker et al, "MAHA: A Program for Datapath Synthesis", Proceedings of the 23rd DAC (Design Automation Conference), Las Vegas, July 1986, pp. 461-466.
3. N. Park and A.C. Parker, "Synthesis of Optimal Clocking Schemes", Proceedings of the 22nd DAC, July 1985, pp. 489-495.
4. E.F. Girczyc and J.P. Knight, "An ADA to Standard Cell Hardware Compiler Based on Graph Grammars and Scheduling", Proc. of the IEEE International Conference on Computer Design (ICCD), October 1984, pp. 726-731.
5. P.G. Paulin, J.P. Knight, E.F. Girczyc, "HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis", Proceedings of the 23rd DAC, July 1986, pp.263-270.
6. P.G. Paulin, J.P. Knight, "Extended Design-Space Exploration in Automatic Data Path Synthesis", Proceedings of the 1986 Canadian Conference on VLSI, October 1986, pp. 221-226.
7. T. Blackman et al, "The Silc Silicon Compiler: Language and Features", Proceedings of the IEEE 22nd DAC, June 1985, pp. 232-237.
8. C. Tseng, D.P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems", IEEE Transactions on CAD, July 1986, pp. 379-395.
9. S. Davidson et al, "Some Experiments in Local Microcode Compaction for Horizontal Machines", IEEE Transactions on Computers, July 1981, pp. 460-477.
10. P. Marwedel, "A New Synthesis Algorithm for the MIMOLA Software System", Proceedings of the 23rd DAC, Las Vegas, July 1986, pp. 271-277.
11. S.Y. Kung, H.J. Whitehouse, T. Kailath, "VLSI and Modern Signal Processing", Prentice Hall, 1985, pp.258-264.
12. J. Nestor, D.E. Thomas, "Behavioral Synthesis with Interfaces", Proceedings of the IEEE ICCAD-86 (International Conference on CAD), November 1986, pp. 112-115.
13. B.M. Pangrle, D.D. Gajski, "State Synthesis and Connectivity Binding for Microarchitecture Compilation", Proceedings of the IEEE ICCAD-86, November 1986, pp. 210-213.
14. M.C. McFarland, "BUD: Bottom-Up Design of Digital Systems", Proceedings of the 23rd DAC, Las Vegas, July 1986, pp. 474-479.
15. N. Park, A.C. Parker, "SEHWA: A Program for Synthesis of Pipelines", Proceedings of the 23rd DAC, Las Vegas, July 1986, pp. 454-460.
16. P.G. Paulin, J.P. Knight, "Scheduling and Allocation for Pipelined ASICs", Submitted to the IEEE International Conference on Computer Design (ICCD '87). **Awaiting acceptance.**
17. H.Trickey, "Flamel: A High-Level Hardware Compiler", IEEE Transactions on CAD, March 1987, pp.259-269.

Authorized licensed use limited to: Hochschule Hamm-Lippstadt. Downloaded on May 14,2023 at 18:25:28 UTC from IEEE Xplore. Restrictions apply.