PARAS: System-Level Concurrent Partitioning and Scheduling

Wing Hang Wong and Rajiv Jain
Department of Electrical and Computer Engineering
University of Wisconsin
Madison, WI 53706
http://polya.ece.wisc.edu/~rajiv/home.html

Abstract
Partitioning for the ASIC designs is examined and the interaction between high-level synthesis and partitioning is studied and incorporated in the solution. Four algorithms (called PARAS) which can exploit this interaction by solving the scheduling and partitioning problems concurrently are presented. PARAS maximizes the overall performance of the final design and considers different chip configurations and communication structures. Experiments, conducted with specifications ranging in size from few to hundreds of operations, demonstrate the success of this approach.

1 Introduction

One choice that has gained considerable popularity with the advances in technology for design implementation is the medium of ASICs which offers a low-cost and relatively high-performance solution. Considerable number of ASIC designs start with a hand-crafted register-transfer level design and go all the way down to layout and fabrication. As the design abstraction has increased from schematic to language-based and is migrating towards behavioral specifications, the level at which the ASIC design starts must also be elevated. Gates and flip flops are already being replaced by larger functional blocks such as adders and registers. Larger amounts of functionalities can be placed on an ASIC and this automatically leads towards system-level design from behavioral specification using ASICs.

As the size of the chips and the operating frequencies of the circuits increase, the ASIC designs will have to contend with wiring delays. While the industry is already experimenting with 300MHz processors, the academicians are developing 1GHz processor designs [8]. This coupled with the increase in chip size is making wiring delays more pronounced and comparable to the gate delays. Future ASIC designs must deal with large wiring delays where a signal may need one or more clock cycles for reaching its destination. The large delay problems which occur in multi-chip ASIC designs will manifest themselves in a single-chip ASIC design. In a single-chip ASIC design, the chip may be divided into several regions, each region containing several functional resources and the regions are connected by some communication structures with wiring delays. System-level ASIC design requires partitioning which will map a behavioral specification onto the regions with an objective of minimizing the total execution time of the final design. In this paper, we identify each region as a separate ASIC. The partitioning process is followed by synthesis to get a register-transfer level design [1, 2, 3, 7, 12, 15].

System-level partitioning has a first-order impact on the cost-performance characteristics of the final design making it imperative for partitioning tools to understand the interaction between lower-level design abstractions and partitioning and to incorporate it in the partitioning algorithms in order to produce good solutions. In this paper, the ASIC partitioning problem and its relationship to high-level synthesis is examined. PARAS can handle the interactions across abstraction levels by solving the scheduling and partitioning problem concurrently are developed. It maximizes the overall performance of the final design and considers different resource configurations and communication structures. Experiments, conducted with specifications ranging in size from few to hundreds of operations, support the validity of this approach.

Partitioning must be based on the functionality of each ASIC and the resources available in the ASICs. Partitions which maximize performance for ASICs with different types of resources may be different. This fact is borne out by the experiments (compare Figures 6b and 7). A partitioning algorithm must consider the functionality of the ASIC and, if necessary, generate different partitions for different ASICs.

The issue of communication is very important in partitioning since it can have a major impact on the area and performance of the final design. Communication across different sets of a partition will incur delays and partitioning may in fact deteriorate the performance of a design. If two sets of a partition need to communicate, and they are floorplanned far apart on the chip, they may incur large delays of the order of one or more clock cycles. A partitioning algorithm must consider this delay during the partitioning of the specification in order to localize communication delays. Partitions which maximize performance may be different for different communication delays between the sets of the partition and a good partitioner must consider these delays while partitioning the specification. The communication requirements may determine the number of global busses (or wires) required in the design which directly impacts the area of the design. Some existing system-level partitioners minimize the number of edges cut in an input specification [6]. This objective was developed for partitioning a design at the physical level where each edge corresponds to a physical wire [5]. At the system-level, however, the number of wires required between decomposed parts generally *less* than the number of edges cut. The number of wires required is equal to the maximum number of bits transferred in any one clock cycle. Even if a large number of edges in the input description are cut, so long as these cut-edges lie in different clock cycles and can share wires for data transfer, the number of wires required for transmission will be small. Further, sharing of wires for handling different data can only be done during scheduling and hence partitioning and scheduling must be performed concurrently.

The objective used for partitioning in [12] is to minimize the number of different types of chips needed for synthesizing the design by regularity extraction. The partitioning algorithm in [7] groups similar operations in one set of the

[0]Research supported by NSF (MIPS 9307830).

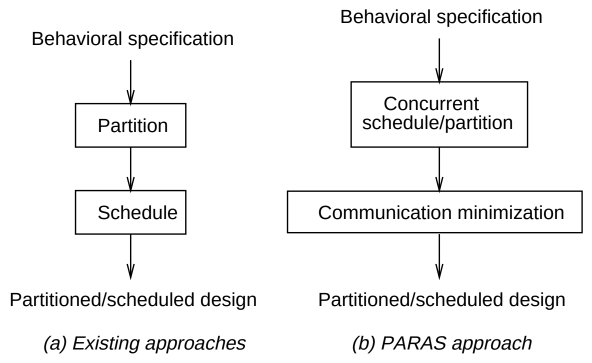
Behavioral specification
Partition
Schedule
Partitioned/scheduled design
(a) Existing approaches
Behavioral specification
Concurrent schedule/partition
Communication minimization
Partitioned/scheduled design
(b) PARAS approach

Figure 1: Two solution approaches

partition since these operations can potentially share resources. Resource sharing can only be done if operations are scheduled over different clock cycles. Forcing like operations to share resources by putting them in different clock cycles may result in a loss in performance due to increased resource contention. Further, resource sharing information is not available to the partitioner until scheduling is done. The inputs to the system by Hung and Parker [3] are a partitioned specification and pin constraints of the chips, and the output is a schedule which considers hardware resources and pins as constraints. Pin constraints are important for multi-chip partitioning. However, as increasing functionality can be added to a chip multi-chip partitioning would have a lesser impact in the future.

The true system-level design objective is to maximize performance while executing a given behavioral specification on a set of cooperating ASICs while the other objectives are an approximation. In this paper, an approach for system-level ASIC design with the objective of maximizing performance is presented.

2 The PARAS Algorithm

In order to use existing partitioners for the system-level ASIC design, the specification has to be first partitioned and then scheduled (Figure 1a). In the proposed approach (Figure 1b), the behavioral specification is partitioned and scheduled simultaneously, after which, communication bandwidth is minimized in an effort to improve the design's performance. In this section, the concurrent partitioner/scheduler and then three algorithms which can minimize communication bandwidth are presented. The complete system is called PARAS (PARtitioner And Scheduler).

2.1 Concurrent Partitioner/Scheduler

PARAS schedules the input specification onto several ASICs each containing several functional units. For example, each ASIC may contain two adders and one multiplier and connected by a one-clock cycle delay one-word communication channel (or bus). Conventional scheduling algorithms do not have the notion of how closely two operations in a specification are connected, which is important when scheduling on several ASICs. For example, in Figure 2 operations *m1* and *m2* are more closely connected as compared to operations *m1* and *m3*. A conventional scheduling algorithm may generate a schedule where *m1* and *m3* are mapped onto one ASIC and *m2* and *m4* onto a different ASIC, thus increasing the communication costs. The notion of close-connection has been defined in several ways, the most common of them being a function of type-overlap and communication requirements [6, 7]. Define $close(o_i, o_j)$ as the delay along the shortest path between operations o_i and o_j (ignoring the direction of the edges). For the AR filter example, $close(m1, m2) = 1$ adder delay and $close(m1, m3) = 5$ adder delays + 2 multiplier delays. The function *close* captures the idea of how soon the values generated by the two operations are required by another operation or how soon a common value is used by these two operations. A small value of *close* implies that the operations are close to each other. Operations *m1* and *m2* are closer than operations *m1* and *m3*. The *close* function can be generated off-line before scheduling. In general, closely connected operations must be placed in the same set of the partition or two sets with small (or zero) communication delay. The *close* function is required for "seed"-ing the schedule and the remaining operations are scheduled in the earliest-start-time fashion. The earliest start-time of an operation on an ASIC is computed by considering the resource contention, precedence relationships and communication bottlenecks, if any. In case of a tie, an ASIC which requires smallest communication cost is selected.

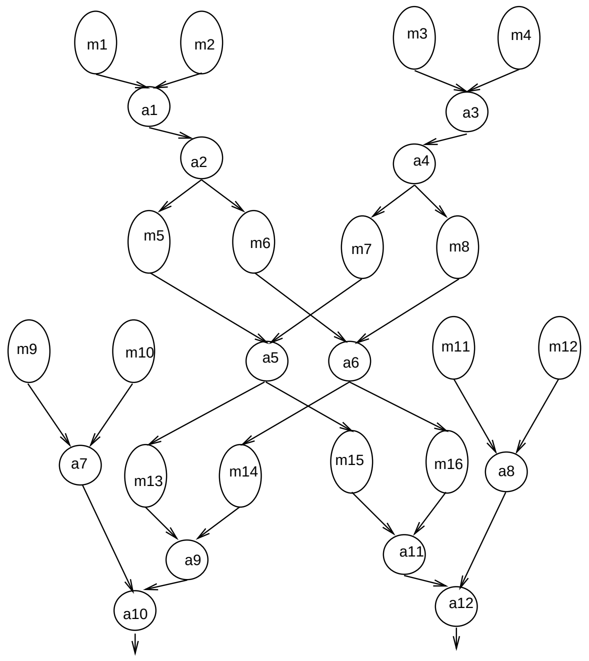

Figure 2: AR Filter data flow graph

The input specifications usually have a "V" structure, implying that the specification has a large number of operations near the input and relatively smaller number of operations near the outputs. Given that among several list-scheduling algorithms, backward and forward list-scheduling algorithms are popular [10], seeding can be either done at the top or the bottom of the specification. Here, we have chosen to do the seeding at the top. The algorithm is presented for two ASICs.

```
Compute ASAP and ALAP values of all operations assuming zero communication delay.
Compute close for every pair of operations.
For $k = 1, \ldots,$ number of resource types in an ASIC {
    1. Backward list schedule using the ALAP values (if identical, then use decreasing ASAP value) with the resource constraint corresponding to type $k$. For other op-
```

eration types, assume unlimited resources.

2. Let the PALAP [14] of an operation be the clock cycle assignment of that operation.

3. Find all operations with minimum ALAP value. Let the set *seeds* contain all these operations.

4. Partition and schedule the seeds first.

a. Randomly choose an operation from *seeds* to be the *primary seed*, and remove it from *seeds*.

b. Put the *primary seed* at the head of *ordered seeds*.

c. While (*seeds* not empty)

Add *seed i* to *ordered seeds* in increasing value of *close(primary seed, seed i)*. (The first half of the operations in *ordered seeds* will be scheduled in one ASIC, and the remaining in the other ASIC.)

5. Partition (but not schedule yet) the non-seed roots.

a. Put all operations which have no predecessor (i.e. ASAP = 1) and which are not in *ordered seeds* in the set *non-seed roots*.

b. Randomly choose an element from *non-seed roots* to be the *primary non-seed root*. Remove it from *non-seed roots* and add it to the set *ordered non-seed roots*.

c. While (*non-seed roots* not empty)

Add *non-seed i* to *ordered non-seed roots* in increasing value of *close(primary non-seed root, non-seed root i)*. (The first half of the operations in *ordered non-seed roots* will be scheduled in one ASIC, and the remaining in the other ASIC. Thus, after one operation from *ordered non seed roots* is scheduled, then if it is in the first half of the list, then all the other operations in the same half will also be scheduled in the same ASIC, and the remaining operations will be scheduled in the other ASIC.)

6. For operation k, let $ISV_k = \sum_{i,j} close(i,j)/$ (no. of immediate successors of k), where i stands for every immediate successor of operation k, and j represents every predecessor [1] of k. ISV is used to resolve the priority during list scheduling in event of a tie. The ISV allows an operation whose average immediate successor is *closer* to its predecessors to have a higher chance of being scheduled in the ASIC in which most of its predecessors are already scheduled in.

7. The forward list-scheduling uses the PALAP as a priority function (in case of a tie, use increasing ALAP – ASAP value; in case of another tie use increasing ISV for selecting operations). The complete designer-imposed resource constraints are used during this step. Observe that concurrent partitioning and scheduling of all operations not in *ordered seeds* is done during this step. First, try to schedule an operation in the earliest clock cycle it can start subject to resource, precedence, and communication constraints. In case of a tie between ASICs, it is scheduled in the ASIC containing more immediate predecessors which complete execution in the previous clock cycle. In case of a tie, it is scheduled in the ASIC containing larger number of its immediate predecessors.

8. Save the current best schedule.

} /* end for */

2.2 Minimizing communication

An important aspect of the system-level ASIC partitioning problem is that a good partition may not satisfy the traditional definition of a partition [5]. Partitioning is traditionally defined as breaking a set into mutually exclusive

[1] Predecessor implies all predecessors, immediate and others.

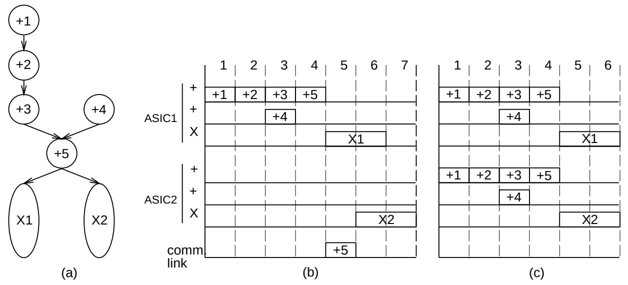

Figure 3: Traditional partitioning may produce inferior designs

and collectively exhaustive subsets. While the collectively exhaustive requirement must be satisfied for functional correctness, the mutually exclusive requirement may produce inferior designs. Consider a small fragment of the elliptic wave filter benchmark shown in Figure 3. Suppose we have two ASICs connected by a one clock cycle delay interconnect and each ASIC contains two adders and one multiplier. Then the best possible design produced by a traditional partitioning tool will require seven clock cycles (Figure 3b). If, however, the mutually exclusive requirement is relaxed and the two subsets may overlap, a design which requires six clock cycles can be obtained (Figure 3c). The overlap trades-off inter-chip communication delay with computation and is not considered by any existing system-level partitioner. We have developed two procedures called *move* and *swap* which try to reduce the communication bandwidth. A third procedure *duplicate* trades-off computation with communication bandwidth. These procedures start from the first clock cycle of the schedule to the last and try to eliminate transfer of a datum from one ASIC to another by moving, swapping or duplicating operations from one ASIC to another. All original inputs are assumed to be available to all the ASICs initially.

Move: The idea behind the *move* procedure is to find a datum that is being transferred via the communication channel from the sending ASIC to the receiving ASIC. If this datum is not used in the sending ASIC, then it may be preferable to generate the datum in the receiving ASIC itself, thus reducing the communication channel usage. To ensure that as a result of eliminating this datum no other datum is added to the communication channel, *move* requires that not only operation i should not have any successors in the sending ASIC, but also that the predecessors of operation i have no successors in the sending ASIC, unless that successor is operation i or a predecessor of operation i. The move procedure also ensures that the receiving ASIC has sufficient free resources so that all operations which need to be moved can be executed in the receiving ASIC without sacrificing performance.

Swap: The basic idea of *swap* is to find an operation i which is executed in clock cycle j in ASIC1 and is not executed in ASIC2 and which generates a datum that is required to be transmitted from ASIC1 to ASIC2. Then, an operation k, which is also executed in clock cycle j, in ASIC2, and which has the same resource type as operation i, and which generates a datum that is required to be transferred from ASIC2 to ASIC1, needs to be found. At the same time, if the predecessor requirements (as described for *move* operation) are satisfied, then the two operations and its predecessors can be swapped (some predecessor operations may be moved and others swapped).

Duplicate: The *duplicate* procedure tries to duplicate an operation from a sending ASIC to a receiving ASIC. Duplicate tries to ensure that the net usage of the communication channel is reduced, and that there exists sufficient resources in the receiving ASIC for executing the duplicated operations. Note that there is no need to check the predecessor requirements as above, since the operations are duplicated and the value is available in both ASICs. Note that as a result of applying the *duplicate* procedure, the communication channel may become available, and hence it may be possible to move some operations to an earlier clock cycle. Thus, after the procedure is successfully executed, the subtree rooted at the operation duplicated is re-scheduled. If an improvement in the performance is obtained then this re-scheduled solution is accepted as the new solution. The *duplicate* concept has been used previously in FPGA mapping as well.

DFG	# of resources/ASIC	# of clock cycles	
		1 ASIC	2 ASICs
AR filter (Figure 2) [12]	1a, 1m	34	19
	1a, 2m	18	14
	1a, 3m	16	14
EW filter (Figure 5) [11]	1a, 1m	28	19
	2a, 1m	21	18
	2a, 2m	18	17
diff. eqn. [11]	1a, 1m	13	9
	1a, 2m	8	7
	1a, 3m	7	6
	2a, 2m	7	7
	2a, 3m	6	6
	1a, 4m	6	6
FIR filter [10]	1a, 1m	18	12
	1a, 2m	15	12
	2a, 2m	11	11
	2a, 3m	10	11
Unfolded EW filter	1a, 1m	84	60
	2a, 2m	52	51
	3a, 2m	50	49
	3a, 3m	49	49
Bandpass Filter [9]	1a, 1m	25	15
	1a, 2m	17	13
	2a, 2m	13	11
	2a, 3m	11	11
	2a, 4m	10	11
Biquad Filter (*) [9]	1a, 1m	27	17
	1a, 2m	19	15
	2a, 2m	17	13
	2a, 3m	14	13
	3a, 3m	13	12
	3a, 5m	11	11
64-point FFT (*) [4]	1a, 1m	576	288
	2a, 2m	288	144
	5a, 5m	117	60
	5a, 10m	60	44
	10a, 15m	39	42
	20a, 20m	30	39

a: adder; m: multipliers
(*) Complex multiplication takes 3 clock cycles.

Table 1: Results

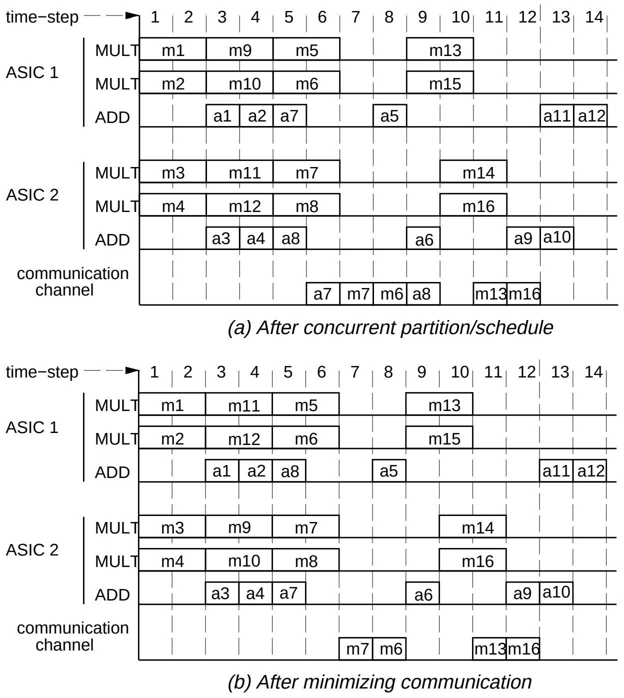

Figure 4: A schedule for AR filter DFG

3 Evaluation of PARAS

For evaluation of the PARAS algorithm several benchmarks ranging in size from about ten operations to about four hundred operations were used. The results (Table 1) show different specifications and the performance of the schedules obtained for different numbers of ASICs. The multiplier and adder requires two and one clock cycle for execution respectively, and a one clock-cycle-delay communication channel is assumed. The first result for the AR filter shows that using one ASIC containing one adder and one multiplier a design requiring 34 clock cycles for execution was obtained. Using two ASICs with one communication channel, PARAS was able to schedule the same specification in 19 clock cycles.

For the AR filter example (Figure 2) with two ASICs each containing two multipliers and one adder connected by a one clock cycle delay interconnect, PARAS obtained the schedule shown in Figure 4a after the concurrent partition/schedule. After minimizing communication, the schedule shown in Figure 4b which requires fewer data values to be transferred is generated. The operations in the two subsets define the partition. The optimal solution to the scheduling problem *without* communication delay using two multipliers and one adder is 18 clock cycles (an upper-bound to the solution) and the optimal solution without communication delay and using four multipliers and two adders is 11 clock cycles (a lower-bound to the solution).

The advantages of the *move* and *swap* procedures can be highlighted from this result. In the schedule shown in Figure 4, $a7$ is executed in ASIC1 in clock cycle five and its output is transferred from ASIC1 to ASIC2 in clock cycle six. $a8$ is executed in ASIC2 in clock cycle five and its output is transferred from ASIC2 to ASIC1 in clock cycle nine. From the graph we see that $a7$ does not have any successor in ASIC1; it has one successor $a10$ in ASIC2. $a8$ does not have any successor in ASIC2; it has one successor $a12$ in ASIC2. Moreover, since $a7$ has two predecessors, $m9$ and

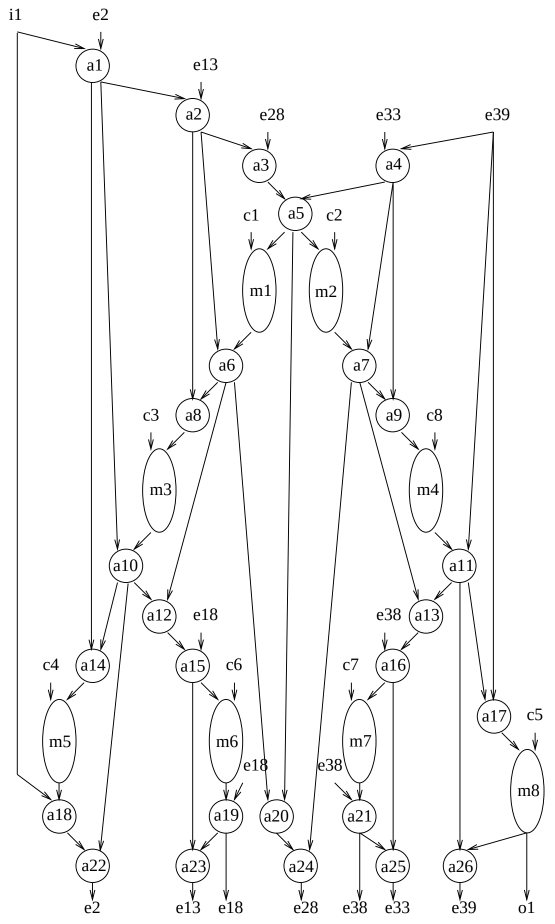

Figure 5: EW Filter data flow graph

$m10$, which both start execution in clock cycle three, and since $a8$ also has two predecessors, $m11$ and $m12$, which also both start execution in clock cycle three, *swap* is successful. As a consequence of the swap, there is no need to transfer either $a7$ or $a8$ on the communication channel anymore. However, in this case, the successful swap did not reduce the total execution time of the schedule; it only reduced the communication bandwidth.

For EW filter (Figure 5) two ASICs each containing two adders and one multiplier were used. [2] After the concurrent partition/schedule program, we obtain the schedule shown in Figure 6a. In this schedule, both $m1$ and $m2$ are immediate successors of $a5$, so if $a5$ is executed in only one ASIC, as in the original schedule, then either $m1$ or $m2$ should be executed after the other one. Since $a5$ has four predecessors, $a1$, $a2$, $a3$, and $a4$, and according to Figure 6a, ASIC2 has enough resources to accommodate all these five operations, these five operations are *duplicated* to obtain the final schedule shown in Figure 6b. As a result of this, $m2$ can now start execution in clock cycle five instead of clock cycle six, which in turn reduces the total execution time of the specification from 19 clock cycles to 18 clock cycles.

Comparison of results to others is hard to make since the PARAS's objective is to maximize performance for a fixed ASIC configuration and input specification. The objective

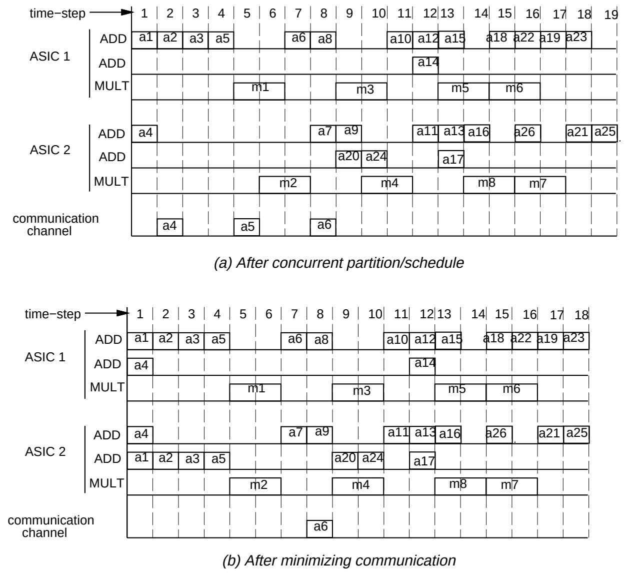

Figure 6: A schedule for EW filter DFG

in the other approaches are different such as minimizing the number of edges cut, minimizing the number of different types of patterns, or clustering similar operation types. Nevertheless, comparison with the result in [6] for the EW filter example is made. In [6], zero communication delay is assumed and the schedule using two ASICs, each containing one adder and one multiplier, requires 19 clock cycles. If, PARAS assumes zero interconnection delay, then a schedule with same performance and equal number of data transfers is generated. If the interconnect has a delay of one clock cycle, then the partition in [6] requires 21 clock cycles while the PARAS solution still requires 19 clock cycles.

The need for using ISV can be explained using the elliptic wave filter. After $a1, a2, a3$ and $a5$ have been scheduled in ASIC1, $m1$ must be scheduled before $m2$. By scheduling $m1$ before $m2$, the scheduler will place $m1$ in ASIC1 also, which in-turn will allow the successors of $m1$ to be scheduled in the same ASIC, thus reducing the communication requirements. The PALAP priority alone cannot handle this, since $m1$ and $m2$ have identical priority and the scheduler may arbitrarily select any one to be scheduled before the other. The ISV will ensure that $m1$ is selected for scheduling before $m2$.

A solution to the AR filter example using two ASICs, each ASIC containing two adders and four multipliers requires 14 clock cycles for execution; a design using one ASIC requires only 11 clock cycles. This illustrates that an input specification must only be partitioned if it will result in a better design. For example, partitioning an input specification into two smaller graphs will result in a bad design if enough resources are available on one chip to execute the entire specification efficiently. This observation is further supported by the results for the 64-pt FFT example (Table 1). With (10a, 15m) resources per ASIC, the design with two ASICs is slower than the design with one ASIC. Increasing the number of resources per ASIC beyond a certain threshold can slow down a partitioned design with communication cost. This fact has not been considered by existing partitioning systems which partition the input specification whether partitioning is desirable or not. The breakpoint depends upon the specification and the number of resources per ASIC. A partitioner must partition only when it will

[2] Determination of what ASICs should contain can be done using estimation techniques [13].

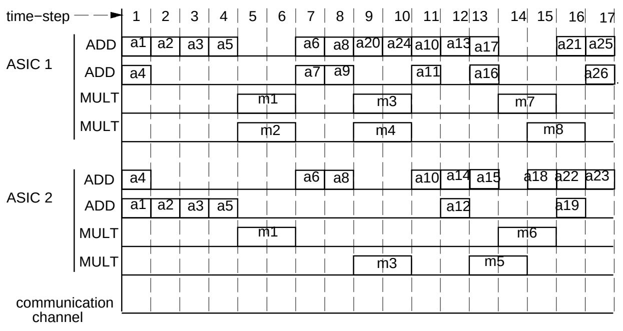

Figure 7: A partition with zero communication

help improve the design, and not otherwise. The PARAS algorithm assumes that partitioning is to be performed and proceeds to seed the ASICs with operations which are not close to each other. Seeding is in direct contrast to the functions of a traditional scheduler where all operations are mapped onto one ASIC. Since the solution techniques of the PARAS and traditional scheduling algorithms conflict, we use an alternate strategy for determining if partitioning is really better than just ordinary scheduling. After an initial schedule/partition is obtained, a check is made to determine if one ASIC is sufficient or not. If an additional ASIC cannot reduce the execution time then an extra ASIC is redundant and must not be used. The first check simply counts the resource utilization for each clock cycle and resource type. If the number of resources used in both ASICs in a clock cycle is less than or equal to the number of resources available in a specified ASIC, then we can collapse the two ASICs into one for that clock cycle. If we can do this for all clock cycles and all resource types, then we can eliminate one ASIC. The second technique for elimination, is to simply schedule the specification using one ASIC and if there is no performance loss then we can dispense with partitioning the specification.

A schedule for the EW filter with each ASIC containing two adders and two multipliers is shown in Figure 7. The schedule, as a result of duplication, has zero communication across the two partitions. This shows that overlapping subsets of the partition may help eliminate communication completely. A traditional partitioner cannot obtain a solution of this type, since some edges will show-up as being cut. Comparing the two schedules for the EW filter in Figures 6 and 7 highlight that the two partitions are different. This implies that depending upon the resource configuration of the ASICs, different partitions may be required for good designs, an idea which does not exist in current partitioning systems.

The PARAS algorithm required about 7,000 lines of C++ code. The run-time for the EW filter example was under 1 second on a SUN SPARCstation 5/70 with 32Mb main memory.

4 Conclusion

To summarize, we have presented a new approach to the partitioning problem which occurs in system-level design. Our solution technique is simple and efficient, allows several tradeoffs to be explored, and has tremendous potential for furthering research in the area of board and system level design. We are currently extending the algorithm to handle conditional branches and loops. Open problems include the development of better and more accurate models of the ASICs and fast and accurate estimation techniques which will help the design tools to explore design space efficiently.

References

[1] R. Gupta, C. N. Coelho Jr., and G. De Micheli. Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1992.

[2] J. Henkel, U. Holtmann, and T. Benner. Adaptation of Partitioning and High-Level Synthesis in Hardware/Software Co-Synthesis. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided-Design*, 1994.

[3] Y.-H. Hung and A. C. Parker. High-Level Synthesis with Pin Constraints for Multiple-Chip Designs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1992.

[4] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill Publishing Company, 1984.

[5] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49, February 1970.

[6] E. D. Lagnese and D. E. Thomas. Architectural Partitioning for System Level Design. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1989.

[7] M. C. McFarland and T. J. Kowalski. Incorporating Bottom-Up Design into Hardware Synthesis. *IEEE Transactions on Computer-Aided-Design*, 9(9), September 1990.

[8] T. Mudge. Designing High Performance Microprocessors. From a talk given at the Department of Electrical and Computer Engineering, University of Wisconsin, April 1995.

[9] C. A. Papachristou and H. Konuk. A High-Level Synthesis Technique Based on Linear Programming. Technical report, Computer Engineering and Science Department, Case Western Reserve University, November 1989.

[10] N. Park and A. C. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Transactions on Computer-Aided-Design*, 7(3), March 1988.

[11] P. G. Paulin and J. P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASIC's. *IEEE Transactions on Computer-Aided-Design*, 8(6), June 1989.

[12] D. S. Rao and F. Kurdahi. Partitioning by Regularity Extraction. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1992.

[13] M. Rim and R. Jain. Lower-Bound Performance Estimation for The High-Level Synthesis Scheduling Problem. *IEEE Transactions on Computer-Aided-Design*, 13(4), April 1994.

[14] M. Rim and R. Jain. RECALS II: A New List Scheduling Algorithm. In *Proceedings of the IEEE International Conference on Acoustic, Speech and Signal Processing*, 1994.

[15] F. Vahid and D. D. Gajski. Specification Partitioning for System Design. In *Proceedings of the ACM/IEEE Design Automation Conference*, 1992.