

Variable Partitioning and Scheduling of Multiple Memory Architectures for DSP

Qingfeng Zhuge, Bin Xiao, Edwin H.-M. Sha
University of Texas at Dallas
Department of Computer Science
Richardson, Texas 75083

Abstract

Multiple memory module architecture enjoys higher memory access bandwidth and thus higher performance. Two key problems in gaining high performance in this kind of architecture are variable partitioning and scheduling. However, there's little research work that has been done on these problems.

In this paper, we present a new graph model for tackling the variable partitioning problem, namely, Variable Independence Graph (VIG), which provides more precise information for variable partitioning compared to the previous graph models. We also present a scheduling algorithm that takes advantages of multiple memory modules, Rotation Scheduling with Variable Re-partition (RSVR). It's a new scheduling technique based on retiming and software pipelining. It may re-partition the variables if necessary during the scheduling process. The experiment results show that the average improvement on schedule length by using the algorithm is 44.8%.

Another major contribution of this paper is that we invent an algorithm for design space exploration on multiple memory architecture. It produces more feasible solutions on a set of schedule length requirement. And our solution have less functional units than Interference Graph model.

1 Introduction

The growing gap of speed between CPU and memory becomes one of the most critical problems for designing high-performance systems. To improve the overall performance, some of the most advanced DSP processors are equipped with on-chip dual data-memory banks accessible in parallel, such as Motorola 56000 [1] and Analog Devices ADSP-2106x series [2]. Since data (variables in the programs) can be partitioned and allocated to separate data-memory banks, and can be accessed simultaneously, the multiple memory module architecture actually offers potentially higher memory bandwidth, and thus potentially higher performance.

This special feature makes the technique of variable mapping and scheduling one of the most important factors in performance optimization of multiple memory module architectures. Whereas, there's little research work has been done to in this area.

Three problems are discussed in this paper and an integral set of solutions are presented: 1. What is the minimum number of memory modules and ALUs to satisfy the required performance? 2. What is the best scheduling and variable partitioning technique for DSP applications that can fully utilize the memory modules and ALUs? And these techniques must be able to deal with the graphs with cycles. 3. What is the best model to represent the variable partitioning problems?

The variable partitioning problem is NP-complete. The difficulty lies in the preciseness of the prediction of parallel memory access demands may appear in a schedule. A too coarse picture of potential memory access parallelism with many unrealizable parallelism may degrade the partition result, and give inferior schedule. On the other hand, a very precise picture may only benefit some specific scheduler and not help much for others. So, a good graph model is very important in providing correct information for variable partitioning.

Special scheduling technique must be designed for a multiple memory module architecture. Based on the nature of variable partitioning problem, a scheduler need to utilize the partition information, and at the same time, may need to adjust the partition on-the-fly to produce an optimal schedule. So the capacity of the performance gaining for multiple memory module architectures are affected by both partitioning strategy and scheduling technique.

The example in Figure 1 shows the *Data Flow Graph* (DFG) of a simple program. There're only three array variables in this program. But two different partition will result in schedules with different lengths, as shown in Figure 2 (a) and (b).

Some previous work on the variable partitioning problem is to transform the partitioning problem to an *interference graph* (IG) [3] [4]. But for most applications in our exper-

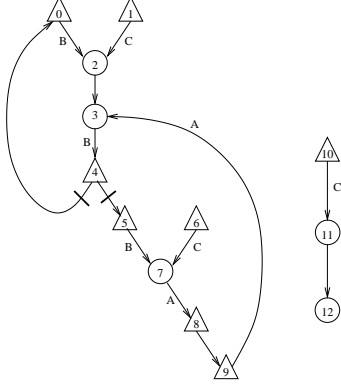


Figure 1. DFG of an example.

CS	A0	A1	M0	M1
0	-	-	5	6
1	7	-	1	0
2	2	-	8	-
3	-	-	10	-
4	11	-	9	-
5	3	12	-	-
6	-	-	-	4

Partition 1 = { A, C }
Partition 2 = { B }

(a)

CS	A0	A1	M0	M1
0	-	-	5	6
1	7	-	0	1
2	2	-	8	10
3	11	-	9	-
4	3	12	-	-
5	-	-	4	-

Partition 1 = { A, B }
Partition 2 = { C }

(b)

Figure 2. (a) Schedule with Partitions of A,C and B; (b) Schedule with Partitions of A,B and C.

iments, the variable partitioning result based on IG model can hardly make any improvement on schedule length. One of the most significant problems of IG is that it can only be applied to *directed acyclic graph* (DAG), which is a limited case in DSP applications. The other problem of IG is that it's not sophisticated enough to reveal the true picture of potentially parallel memory accesses that can really be exploited by a scheduler. Other variable partitioning techniques in previous works are restricted to some specific architecture [5]. So the models presented in this kind of work cannot be applied to a general multiple memory module architecture.

In this paper, we introduce a new graph model for variable partitioning problem, that is, *Variable Independence Graph* (VIG) model. This model reflects all the potentially parallel memory accesses that may actually occur in

scheduling. It also gives a priority metric for different kind of parallel access demands. The graph is constructed from *Data Flow Graph* which may contain cycles with delay. For the scheduling problem, we invented a new scheduling algorithm, *Rotation Scheduling with Variable Re-Partition* (RSVR), to produce compact schedules. RSVR is based on retiming and software pipelining. It may dynamically re-partition the array variables in the procedure of scheduling if necessary. The experiment results on a variety of applications show that constant improvements are achieved on schedule lengths by using our algorithms. The average improvement by using VIG model is 32.1%. The improvement by using RSVR is up to 52.9%. An average improvement of 44.8% is achieved from our experiments. To the authors' knowledge, RSVR produces the most compact schedules for multi-memory module DSP architectures.

Although a schedule can be produced based on a certain partition, the effect of various number of memory modules on the schedule length is not clear. To study the effect of multiple memory modules on the design space, we invent an algorithm to find the minimum number of functional units for a required schedule length. The experiment results show that our approach produce feasible solution for all the required schedule lengths, while the Interference Graph model can only give solution in half of these cases. And our solution gives less number of functional units compared with these from Interference Graph model.

2 Definitions and Properties

For a DSP equipped with multiple ALUs and multiple memory modules, the variable partitioning problem is to map the variables of a program to different memory modules, so that, memory operations on the different memory modules can be executed simultaneously. It is obvious that some partition results allow more parallel memory accesses to be implemented in scheduling. And this may help a scheduler to produce a shorter schedule.

To solve a variable partitioning problem, first we need to identify all the potential parallel relationships between array variables, but at the same time exclude the parallelism that cannot be realized by scheduler. To maximize the benefits of parallel memory accesses, we need to choose proper priority metric for all the potential parallel relationships. And also the partition result should be able to give scheduler better chances to produces a more compact schedule.

Many media and DSP application programs can be represented by a *Data Flow Graph* (DFG). It reflects the data dependencies of both ALU operations and memory operations inside iteration or between iterations.

Definition 2.1 A *Data Flow Graph* (DFG) is a directed weighted graph $G_D = \langle V, E, d, t, S \rangle$ where V is a set

of computation nodes that represent ALU operations and memory operations (load or store), E is the edge set which defines the precedence relations from nodes in V to nodes in V , $d(e)$ represents the number of delays for an edge e , $t(e)$ represents the computation time of a node v , $S(e)$ represents the variable accessed by a node v on edge e ($u \rightarrow v$).

An edge ending at a memory operation node represents a Store; an edge starting from a memory operation node represents a Load. An edge label $S(e)$ indicates the name of a variable that is accessed by edge e . An edge e from u to v with $d(e)$ delay reflects precedence relationship between different iterations. A static schedule must obey the precedence relations defined by the subgraph consisting of edges without delays in a DFG. It's easy to see that memory operation nodes that are not precedent to each other are candidates for potential parallel accesses.

The idea here is to extract and translate the appropriate parallelism information into a graph model, which we called a *Variable Independence Graph*.

Definition 2.2 A *Variable Independence Graph (VIG)* is an undirected weighted graph $G_V = \langle V, E, w \rangle$ where V is a node set represents variables, E is an edge set that indicates there is no dependence between the two nodes of an edge $e(u, v)$, i.e., there is potential parallelism between them. $w(e)$ represents the force or extent of independence between the two nodes of edge $e(u, v)$.

We are going to present several constructions of VIG, and shows that the last one, VIG-Best, is the best. We first introduce *access set* as follows:

Definition 2.3 Given a DFG $G_D = \langle V, E, d, t, S \rangle$, the *Access Set* of variable X , $ACCESS(X)$, is the set of memory operation nodes that access X .

By removing the edges with delays of a given DFG, DAG part is remained. It should be clear that only sibling memory operation nodes in this graph are possible to be executed in parallel.

Definition 2.4 The *First Build: VIG-1* For any two different variables X_i and X_j , if there exists a pair of sibling nodes from $ACCESS(X_i)$ and $ACCESS(X_j)$, there is an independence edge (X_i, X_j) .

Although there can be plenty of parallel demands, not all the parallelism can be exploited in scheduling. And the redundant edges may lead to inferior partition result. We choose *mobility window* (MW) to assist the eliminating of these edges, since two nodes are impossible to be scheduled in the same control step, if their mobility windows are not overlap.

Definition 2.5 Given a DFG $G_D = \langle V, E, d, t, S \rangle$, The *Mobility Window* of some node v in DFG, $MW(v)$, is a series of control steps starting from the position of v scheduled by ASAP scheduling, ending with that scheduled by ALAP scheduling.

Definition 2.6 The *Second Build: VIG-2* For any two different variables X_i and X_j , if there exists a pair of sibling nodes from $ACCESS(X_i)$ and $ACCESS(X_j)$, and $MW(u) \wedge MW(v) \neq \emptyset$, there is an independence edge (X_i, X_j) .

Although VIG-2 identifies all parallel demands, it does not contain any information about the priority of the independence relationship. All edges in the VIG-2 have the same priority. To achieve the most benefits of memory access parallelism, some kind of parallel demands are preferred than others.

Definition 2.7 The *Third Build: VIG-Best* For any two different variables X_i and X_j , if there exists a pair of sibling nodes u and v from $ACCESS(X_i)$ and $ACCESS(X_j)$, and $MW(u) \wedge MW(v) \neq \emptyset$, there is an independence edge (X_i, X_j) in Variable Independence Graph. For every pair of nodes u and v , there is a possibility that u and v may appear in the same control step within their mobility windows. And the edge weight $w(X_i, X_j)$ is the sum of the possibilities for all pairs of sibling nodes of $ACCESS(X_i)$ and $ACCESS(X_j)$.

The example in Figure 3 shows that independence edge (A, B) have higher priority than edge (A, C) and (B, C) . So variables A and B need to be allocated to different variable partitions. While the interference graph cannot provide this information for partitioning, a sub-optimal partition result may be produced.

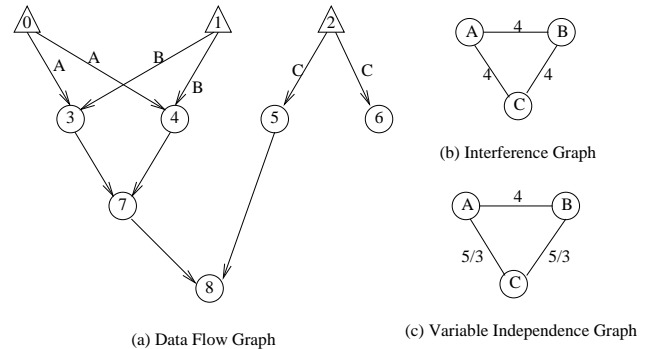


Figure 3. Comparison of Interference Graph and VIG-Best.

Figure 4 is the Variable Independence Graph of our example. It shows that $w(B, C) > w(A, C) > w(A, B)$.

So the good partition on a dual memory bank architecture is $Partition_1 = \{A, B\}$ and $Partition_2 = \{C\}$. The schedule produced with this partition result gains 1 less step than that the inferior partition. Both schedules are shown in Figure 2.

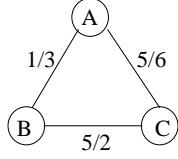


Figure 4. The Variable Independence Graph of the first example.

Based on the previous definition of Variable Independence Graph, the best variable partition is achieved if the total weight of edges that cross the different partitions is maximal.

3 Algorithms

In this section, we describe our major algorithms for variable partitioning and scheduling on multiple memory module architectures. Our approach is to produce the initial partition by applying variable partitioning algorithm on Variable Independence Graph. Then we schedule the DAG part of the DFG based on this initial partition by using list scheduling algorithm. In order to achieve a more compact schedule, we use our newly developed DFG scheduling algorithm, *Rotation Scheduling with Variable Re-partitioning*, RSVR, to iteratively compact the schedule while re-partitioning the variables. And finally, we present a new design space exploration algorithm by using RSVR.

Variable partitioning algorithm uses greedy strategy to partition the nodes of the input Variable Independence Graph into multiple disjoint node sets. Each node set corresponds to a data-memory module. We select a node from input graph to be partitioned. The weights of the edges that between different node sets are calculated and compared when we try to put this node into a node set. We always choose a node that creates the largest edge weight gains among the node sets. And this node is put into the correspondent node set. The algorithm terminates when all the nodes in the input graph are partitioned.

The RSVR algorithm, Rotation Scheduling with Variable Re-partitioning, is based on the concept of retiming and software pipelining. It is designed to produce a more compact schedule iteratively [7]. In this algorithm, the implicit retiming operations moves the delays in the original DFG, and so modifies the DAG. The retiming effects will be discussed later. For the sake of convenience, we apply

rotation only on the first row of nodes in the schedule. Actually, rotation scheduling can be conducted with different size of rotatable sets and different range of rotation phases [7]. And these parameters can be set up for the rotation cycle of RSVR algorithm to explore more rotation opportunities.

By using this algorithm, the variables can be re-partitioned to a different memory module as long as this assignment helps to compact the schedule length. If the scheduling is not increased after variable re-partitioning, rotation scheduling [7] can be continued. The advantage of this algorithm is that the variable partitions can be adjusted during software pipelining. Rather than a static partitioning result, RSVR provides a more flexible way to modify the partitions and to benefit some specific scheduling requests.

Algorithm 1 Procedure of Rotation Scheduling with Variable Re-partitioning

Input: DFG G_D ; initial schedule S ; variable partitions P_1, P_2, \dots, P_n
Output: New schedule S_r ; new partitions P_1, P_2, \dots, P_n
for all $i = 0, \dots, S.length$ **do**
 Rotate the first row of schedule S ;
 Construct DAG G_A of DFG G_D ;
 if The new schedule $S_r.length < S.length$ **then**
 $S = S_r$;
 Continue the rotation;
 else
 Get the memory operation v that access variable X of memory module M_k to be re-partitioned;
 for all The other memory modules **do**
 Select a module M_j that has enough space to reside all the memory operations that access X , and has no precedence violation in G_A after moving these nodes;
 $P_k = P_k - \{v\}$; $P_j = P_j + \{v\}$;
 Continue the rotation;
 end for
 end if
end for
Return $S_r, P_1, P_2, \dots, P_n$;

We use RSVR as a core algorithm in our design space exploration algorithm. Given a required schedule length and a DFG, the algorithm produces the minimum number of functional units required to meet the target schedule length. We start the process with *As Soon As Possible (ASAP) Scheduling*. The upper bound of the number of functional units is produced from the ASAP schedule. Then, we apply list scheduling on the DAG part of DFG to get an initial schedule. If the initial schedule length is less than the target schedule length, the algorithm goes to its first branch. It reduces the number of functional units by 1, and test the

resulted schedule length. If it's acceptable, the algorithm continues the reduction of the number of functional units. Otherwise, It uses RSVR algorithm to compact the schedule length and re-partitions the array variables at the same time if necessary. If we get a new schedule length that is less or equal to the target schedule length. The algorithm reports the number of functional units, the new schedule and schedule length, and stops. The reduction of functional units can be applied to both ALUs and memory modules depending on the designer's input. If the initial ASAP schedule cannot meet the target schedule length, the algorithm goes to its second branch. The second branch starts with retiming on the input DFG. The *Clock-Period Minimization Algorithm* [6] is used to get a new DFG. If the critical path is still greater than the target schedule length. The algorithm reports error and exits. If the critical path after retiming meets the target, it goes to execute the first branch, until a feasible solution is obtained.

4 Experiments and Discussion

Applications	IG	RSVR	Improve
IIR	17	8	52.9%
Diff	21	11	52.4%
All-pole	29	18	37.9%
4-stage Lattice	44	22	50.0%
Elliptical	35	24	31.4%
Volterra	41	23	43.9%

Table 1. The improvements of Rotation Scheduling with Variable Re-partitioning algorithm.

We have experimented with our approaches on a variety of filter applications which are usually used in DSPs, and the results are very promising.

The resulted schedule lengths of our method, that is, Rotation Scheduling with Variable Re-partitioning Algorithm with partitions produced by *VIG-Best* model, are compared against the schedule lengths produced by list scheduling with partitions of *Interference Graph* model [3]. Because of the space limitation, we only show the DFG and graph models for IIR filter in Figure 5 and Figure 6. Variable partitioning process may produce different results from the two graph models. We generate the schedules for both of these two partitioning results on a simulated processor with two ALUs and two on-chip memory modules. The results are shown in Table 1. The experiment results show that significant improvement is made by applying our integral approach. The largest improvement of our approach is 52.9%. The average improvement is 44.8%.

Although some advanced DSPs are equipped by dual

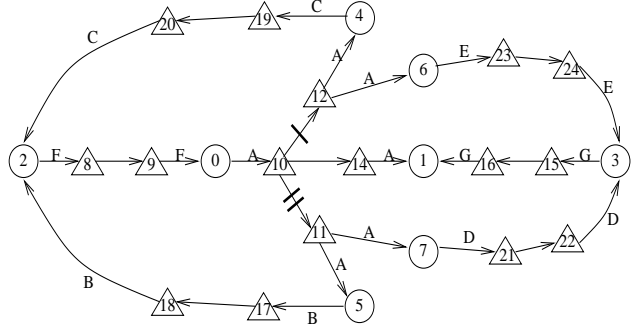


Figure 5. Data Flow Graph of IIR filter.

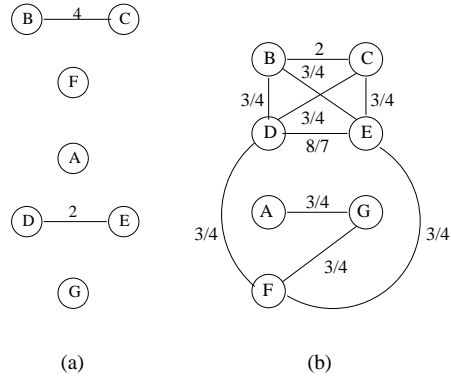


Figure 6. (a) Interference Graph of IIR filter; (b) Variable Independence Graph of IIR filter.

data-memory banks to obtain higher performance, the effect of more memory modules on performance is not clear. In the following experiments, we first show the improvement of RSVR algorithm by comparing our schedule lengths with those produced by list scheduling with the same number of ALUs and Memory modules. Then we explore the design space of the number of functional units under a set of required schedule lengths.

To study the influence of the number of data-memory modules on the schedule length, we do experiments on the same set of applications by simulating processors equipped with three or more memory modules. The resulted schedule lengths by applying List Scheduling (LS) and Rotation Scheduling with Variable Re-partitioning (RSVR) are list side by side in Table 2. To show the effects of software pipelining in RSVR, we doubled the delays in all the cases. The experiment results show that the increasing number of memory modules helps RSVR make shorter schedules, but it does not help much for list scheduling method. So it is clear to see that without a good scheduling algorithm, the effectiveness of using an architecture with more memory

Application	1A1M		2A1M		1A2M		2A2M		2A3M		2A4M	
	LS	RSVR	LS	RSVR	LS	RSVR	LS	RSVR	LS	RSVR	LS	RSVR
IIR Filter	17	16	17	16	13	9	12	8	12	7	12	6
Differential Equation	21	21	21	21	13	11	11	11	11	8	11	7
All-pole Filter	29	29	29	29	27	20	27	18	27	11	27	9
4-stage Lattice Filter	44	44	44	44	27	26	23	22	19	16	19	14
Elliptical Filter	42	41	35	35	40	36	28	24	28	21	28	17
Voltera Filter	41	38	41	37	30	27	29	23	29	17	29	15

Table 2. Schedule lengths obtained by list scheduling and rotation scheduling with variable re-partitioning for various functional units.

Application	Target Schedule Lengths										
	45		35		25		20		15		
	IG	Ours	IG	Ours	IG	Ours	IG	Ours	IG	Ours	
IIR Filter	1A1M	1A1M	1A1M	1A1M	1A1M	1A1M	1A1M	1A1M	1A1M	X	1A2M
Differential Equation	1A1M	1A1M	1A1M	1A1M	1A1M	1A1M	1A1M	X	1A2M	X	1A2M
All-pole Filter	1A1M	1A1M	1A1M	1A1M	X	1A2M	X	1A2M	X	2A3M	2A3M
4-stage Lattice Filter	1A1M	1A1M	X	1A2M	X	2A2M	X	2A3M	X	2A4M	2A4M
Elliptical Filter	1A1M	1A1M	2A2M	2A1M	X	2A2M	X	2A4M	X	3A4M	3A4M
Voltera Filter	1A1M	1A1M	X	1A2M	X	2A2M	X	2A3M	X	2A4M	2A4M

Table 3. The minimum number of functional units for required schedule lengths; “A”– ALU, “M”– Memory Module; “X” means no feasible solution.

modules cannot be fully exploited.

In the following set of experiments, we use our design space exploration algorithm to produce the minimum number of functional units for a feasible schedule length under the schedule length requirement. And our solutions are compared with the those produced by list scheduling with variable partitions based on Interference Graph model under the same schedule length requirement. As shown in Table 3, our algorithm gives feasible solutions for all the schedule length requirements for all the applications, while the approach based on Interference Graph model only gives feasible solutions in less than half cases. And for the same schedule length requirement, our algorithm gives a solution with less or equal number of functional units. The experiment results show that our design space exploration algorithm, which takes full advantage of software pipelining and the architectural feature of multiple memory modules, is very effective in exploring the required number of functional units in a multiple memory module architecture.

Retiming effect on Variable Partitioning

The retiming moves the delay on DFG edges, and changes the inter-iteration dependencies [6] [7]. It affects the potential parallelism among variables. The *Variable Independence Graph* will be changed after retiming, and so the variable partition results. The Variable Independence

Graphs before and after retiming are shown in Figure 7. In this case, retiming increases the memory access parallelism, since more memory operation nodes from different iterations are now pipelined in one iteration. The effect is shown by more edges in the corresponding VIG in Figure 7(b). Our RSVR algorithm utilizes the effect of retiming to compact the schedule.

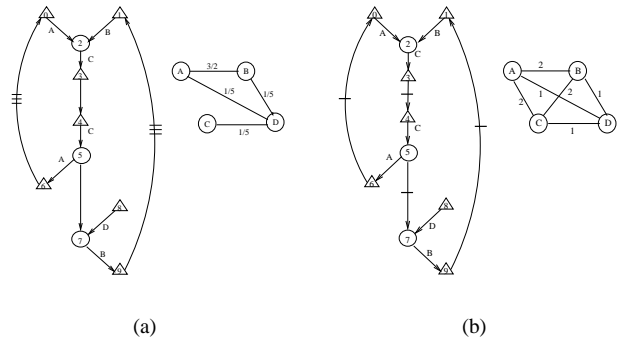


Figure 7. (a) DFG and VIG before retiming; (b) DFG and VIG after retiming.

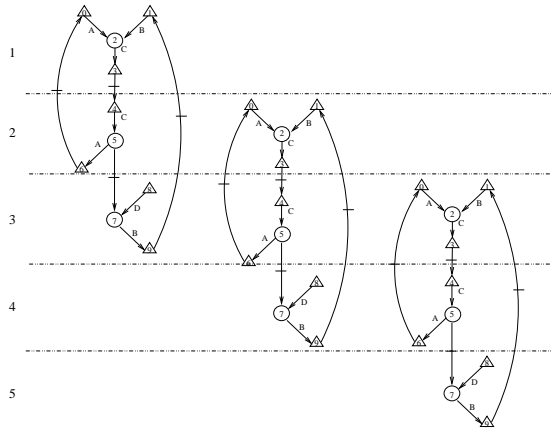


Figure 8. Reorganized iterations after retiming.

5 Conclusion

Traditional scheduling techniques are not effective for making compact schedules in an architecture with multiple memory modules. We presented a theoretical graph model, *Variable Independence Graph*, for variable partitioning problem. Compared to the previous models [3] [4], it can handle the DFGs with cycles and delays, and it provides more complete and precise potential parallelism demands for variable partitioning.

We also presented a new scheduling technique, *Rotation Scheduling with Variable Re-partitioning* (RSVR), to make compact schedules effectively for multiple memory module architectures. Because performing DFG scheduling with variable re-partitioning at the same time, it produces the best-known results to the authors' knowledge. A new design space exploration approach driven by RSVR is invented to product the minimum number of functional units

for required schedule length. Compared to the Interference Graph model, our approach produces smaller number of functional units and much more feasible solutions under the same schedule length requirement.

References

- [1] Motorola, Inc., *DSP56000 Digital Signal Processor Family Manual*, 1995.
- [2] Analog Devices, Inc., *ADSP-21000 Family Application Handbook Volume 1*, 1994.
- [3] R. Leupers and D. Kotte, "Variable partitioning for dual memory bank dsp's," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 2001, vol. 2, pp. 1121–1124.
- [4] M. A. R. Saghir, P. Chow, and C. G. Lee, "Exploiting dual data-memory banks in digital signal processors," in *Proceedings of the 7th international conference on Architectural support for programming languages and operating systems*. ACM, 1996, pp. 234–243.
- [5] Ashok Sudarsanam and Sharad Malik, "Simultaneous reference allocation in code generation for dual data memory bank asips," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 2, pp. 242–264, 2000.
- [6] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.
- [7] L.-F. Chao, A. S. LaPaugh, and E. H.-M. Sha, "Rotation scheduling: A loop pipelining algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 3, pp. 229–239, 1997.