Combining Speculative Execution and Conditional Resource Sharing to Efficiently Schedule Conditional Behaviors

Apostolos A. Kountouris, Christophe Wolinski
IRISA
Campus Universitaire de Beaulieu
F-35042 Rennes CEDEX
FRANCE

Scheduling conditional behaviors necessitates the use of a variety of scheduling optimization techniques like conditional resource sharing and speculative execution. Previous research work has clearly shown their effectiveness. The developed heuristics have several drawbacks relating to the effects of syntactic variance on the results. In this paper a list-based scheduling heuristic that exploits conditional resource sharing and speculative execution possibilities, is presented. Its results are quite insensitive to syntactic variance and conditional behavior is effectively accounted for by a probabilistic priority function.

1. Introduction

Scheduling behaviors having a complex conditional structure has been thoroughly investigated in previous research work mainly due to the fact that traditional DFG based heuristics are not adequate to efficiently handle this kind of descriptions [1]. Several better adapted heuristics were proposed [1], [2], [3], [4], [5], [6]. The quality of their results, depends heavily on the ability to exploit conditional resource sharing [2], [4], [6], [7] speculative execution possibilities [3], [5], [16], [17], and shorten path lengths using node duplication techniques [3]. In resource constrained scheduling these techniques permit to better utilize the hardware resources in the datapath and obtain better schedules which result in shorter execution paths and less control logic. An important issue, underlined in previous work [9], [10] relates to the effects of the syntactic variance of the input descriptions, on the synthesis results. These negative effects intervene in two distinct but interrelated levels as far as scheduling conditional behaviors is concerned; mutual exclusiveness detection and scheduling. CDFG based mutual exclusiveness detection techniques (i.e. [3], [11]) using the structure of the input description, produce different schedules for semantically equivalent but syntactically different descriptions due to the variability on the amount of the detected mutual exclusiveness [9]. Furthermore, CFG-based scheduling (i.e. PBS in [6]) is very sensitive to the statement order in the input description. From the above discussion it is clear that in order to efficiently schedule conditional behaviors and possibly optimize the resulting controllers, we have to exploit conditional resource sharing [2], [4], [6], [7] and speculative execution [3], [5], [16], [17], shorten path lengths using node duplication techniques [3] and last but not least cope with syntactic variance [9]. This is the objective of the heuristic described in the following sections.

2. The HCDG representation

The HCDG [20] is a special kind of *directed* graph that represents data and control dependencies from a *data-flow* perspective. It consists of the *Conditional Dependency Graph* (CDG) and the *Guard Condition Hierarchy* (GCH).

The CDG consists of a set of nodes and set of edges both labeled by *guard* conditions (called *guards* in the sequel). Guards are a special type of nodes and represent boolean conditions that *guard* the execution of operations and the assignment of values to variables. They have also been used as a formal control model in [4], [5] where a CDFG design representation is used. The rest of nodes correspond to operations (i/o, computation, data merging and storage with either register or transparent latch semantics) that compute/assign values to variables. Edges represent *data* and *control* dependencies of the nodes.

Guards are equivalence classes of the HCDG elements. For instance, nodes labeled by the same guard are active (carry a value) at the same logical instants. The *Guard Condition Hierarchy* (GCH) represents the *inclusion relations* between guards. In [21], it is implemented as a hierarchy of BDD's. Using BDD's two things are achieved; first, redundancy is avoided since *equivalence* between guard formulas can be easily

established, and second, using a factorization process [21] it is easy to find the *maximum* depth in the tree that a guard can be inserted. Control representations based on BDD's have already been used in previous work ([15], [4], [5]). The originality of the GCH lies on the hierarchy construction and not at the use of BDD's which are simply used for their efficiency. Guard inclusion is important in efficiently detecting mutual exclusiveness, independently of syntactic variances. As it was shown in [18], [19] it permits to minimize the number of mutual exclusion tests significantly.

3. HCDG List Scheduling Heuristic

One of the important advantages of *list scheduling* is that its quality depends on the quality of the priority function [1]. The contribution of this work lies in the choice of the priority function and on the policy of scheduling nodes on the available resources at each iteration of the algorithm.

Scheduling policy outline: At each scheduling step nodes that can be scheduled on the available resources are partitioned into the *RDY* and *SPC*, groups corresponding to nodes that can be scheduled *without* and *with* speculative execution. Nodes in *RDY* observe both their data and control dependencies while nodes in *SPC* fully observe only their data and partially their control dependencies (weak dependencies in [17] is a similar concept). For brevity, the former will be called *ready* and the latter *spec* nodes. At each step, *ready* nodes are scheduled first ([17] has a similar view), exploiting conditional resource sharing possibilities. Once no more *ready* nodes can be scheduled, resource utilization is increased by further conditionally sharing used resources with *spec* nodes. Finally, if unused resources still remain we try to utilize them by conditionally sharing *spec* nodes. Each schedulable node is associated to a schedule guard (*sguard*) indicating the condition under which the node will execute if it is scheduled in the current control step. For *ready* nodes the *sguard* is the guard labeling the node in the HCDG; hence it is statically available. For *spec* nodes the *sguard* has to be dynamically computed at each step (similarly to guard smoothing [16], dynamic guards [5], dynamic CV's [3]) and corresponds to the most profound already scheduled guard ancestor of the *spec* node's guard in the guard hierarchy. Using this technique we maximally exploit conditional resource sharing possibilities also for nodes that are speculatively executed.

An adaptive probabilistic priority function: A special priority function was elaborated. For each guard its probability p is computed from their boolean definition formulas. Using a *sum-of-products* definition the probability is given as the sum of products of the corresponding boolean variable probabilities. For comparison operations and input booleans a probability of 0.5 is given to both *true* and *false* outcomes. A similar technique to obtain predicate probabilities was used in [7]. Next, ASAP and ALAP schedules are used to obtain node mobilities when both control, data dependencies are observed ($mob_{c/d}$) and when only data dependencies are observed (mob_d). Node initial weighted urgencies are given by: $wurg = p_s \cdot 1/(1 + mob)$. Depending on the case (*ready* or *spec* node), either $mob_{c/d}$ or mob_d is used; p_s corresponds to the probability of the node scheduling guards (*sguard*s). At the beginning of each scheduling step initial weighted urgencies of *ready* and *spec* nodes are adjusted. Adjustment becomes effective once a node has remained unscheduled for a number of iterations higher than its mobility since it became schedulable for the first time. This is achieved by: $awurg(n) = wurg(n) + a(n) \cdot p_s(n)$ where $a(n)$ is the adjustment coefficient for node n:

$$a(n) = \begin{cases} 0 & \text{if } ((currcstep - ALAP(n)) < 0) \\ 1 + (currcstep - ALAP(n)) & \text{otherwise} \end{cases}$$

A *weighted-projected-resource-demand* (*wprd*) measure was also defined as:

$$wprd(n) = \frac{\sum_{\forall t} |R_t| \cdot \sum_{\forall s \in succ(n,t)} p(h(n,s))}{|R|}$$

where R the set of all available resources with cardinality $|R|$, R_t the set of available resources of type t with cardinality $|R_t|$; $succ(n, t)$ are the successors of node n that will become schedulable for the next control step and can be mapped on a resource of type t; $h(n, s)$ the guard of the dependency from node n to node s and $p(h(n, s))$ its probability. The multi-level priority function for a node n ($pr(n)$) is based on the following sorting keys: $pr_1 = awurg$, $pr_2 = p_s$, $pr_3 = wprd$. The first key (pr_1) accounts for the node mobilities in a conditional execution context. The second key (pr_2) further takes into account the conditional nature of the scheduled behaviors giving higher priority to more frequently executing nodes among nodes of equal urgencies. Finally, the third key (pr_3) provides a local measure of how the available resources will be able to accommodate the resource demands provoked by scheduling a node in the current control step.

Conditional Resource Sharing: For each resource type a guard compatibility graph is constructed called

MEG for *Mutual Exclusiveness Graph* [19]. Each vertex corresponds to a scheduling guard and has an associated list of operation nodes being active under this guard and can be assigned to the resource of that type. Operations in the list are sorted according to their priorities with *ready* nodes sorted before *spec* nodes. MEG edges indicate guard mutual exclusiveness and a *clique* corresponds to a group of pair-wise mutually exclusive guards. A list of nodes that can share a resource is constructed in three steps. First, *ready* nodes are considered; the guard clique containing the higher priority node and a maximum number of other high priority nodes is iteratively constructed. Next, the clique is enlarged with guards of higher priority nodes that are candidates for speculative execution. Finally, for each guard in the clique the first node in it operation list is returned. The best adapted algorithm to find such cliques is based on the *initial-graph-partition* algorithm presented in [13]. Heuristics like [14] are not as well adapted to satisfy our clique construction objectives since clique maximality is not a good optimization objective in our case.

This process has several advantages. The list-scheduling priority criterion is satisfied for the greatest number of distinct execution instances (paths), simultaneously because the constructed clique contains the highest priority node and the largest number of other higher priority nodes that can share a resource with it. In respect to [9] and [5], speculative execution is considered only after normally executing nodes have been scheduled. In this way the risk of lengthening execution paths by displacing normally executing operations in favor of speculatively executing ones, is avoided. Conditional resource sharing is exploited during scheduling and not before and so lengthening of execution paths due to inappropriate conditional resource sharing (i.e. [2], [11]), is also avoided.

4. Experimental results

The HCDG-based list scheduling heuristic exploiting conditional resource sharing and speculative execution was tested on a set of benchmarks appearing in previous related literature. These are: *kim*, *waka*, *maha* and *jian* from [2], [7], [12], [8] respectively. For each benchmark the HCDG was constructed, the guard hierarchy was refined, and the HCDG was transformed so that each node is activated only as often as its results are used in succeeding operations. The guard mutual exclusiveness was established using the techniques described in [18]. The HCDG based list scheduling heuristic is compared to other similar heuristics. The obtained results are given in figure 1 for various resource constraints (one cycle resources) in terms of *total* / *longest* path / *shortest* path numbers of states. Published results of other approaches (i.e. *Kim* [2], *CVLS* [7], [3], *PBS* [6], *Brewer* [5], *ADD-FDLS* [9]), when available for the particular resource constraints, are also given. Our results are at least as good as the best previously published results. Chaining is applied as a post-scheduling optimization and it is worth noting that even without it remarkably good results are obtained.

Resources	Schedule by approach				
	Kim	PBS	crit. path	Brewer	ours
cmp: 0, +: 1, -: 1, cn: 1	8/8/3	-	-	-/5/-	5/5/4
cmp: 0, +: 1, -: 1, cn: 2	6/5/2	9/5/2	8/8/-	-	5/5/4
cmp: 0, +: 2, -: 3, cn: 1	-	-	-	-/4/-	4/4/2
cmp: 0, +: 2, -: 3, cn: 3	3/3/2	-	4/4/-	-	3/3/2
cmp: 0, +: 2, -: 3, cn: 5	-	4/3/1	-	-	3/3/2

(a) Results for the "*maha*" benchmark

Resources	Schedule by approach				
	CVLS	Kim	PBS	Brewer	ours
cmp: 1, +: 1, -: 1, cn: 1	7/7/5	7/7/4	-	-	7/7/4
cmp: 1, +: 1, -: 1, cn: 2	-	7/7/3	8/7/3	-/7/-	6/6/3
cmp: 1, ALU: 2, cn: 1	-	-	-	-	7/7/4
cmp: 1, ALU: 2, cn: 2	-	6/6/3	6/6/3	-	6/6/3

(b) Results for the "*waka*" benchmark

Resources	Schedule by approach			
	Kim	Brewer	ADD	ours
cmp: 2, +: 2, -: 1, cn: 1	8/8/6	-	6/6/5	6/6/6
cmp: 1, +: 2, -: 1, cn: 1	-	-/6/-		6/6/6
cmp: 2, ALU: 2, cn: 1	-	-	-	6/6/6

(c) Results for the "*kim*" benchmark

Resources	Schedule by approach	
	ours (cn=1)	ours (cn=2)
cmp: 1, +: 1, cn: 1	4/4/3	4/4/3
cmp: 1, +: 2, cn: 1	4/4/2	3/3/2

(d) Results for the "*jian*" benchmark

Figure 1. Benchmark comparative scheduling results

The differences in terms of the higher number of states in shortest paths is explained by the fact that usually the total number of states is lower than in other approaches and so resources are better utilized at each state. Finally, the insensitivity of the scheduling results to the effects of syntactic variance was evaluated (table 1). Our heuristic was applied on two semantically equivalent but syntactically different descriptions (descr.1, descr.2), for each benchmark. The first, has a maximal conditional nesting as opposed to second one where conditions are flattened and each assignment is in its own conditional block. The achieved insensitivity can be attributed first to the dataflow nature of the HCDG (results independent of statement order) and second to the guard hierarchy construction and the mutual exclusiveness detection process that yields the same amount of exclusiveness for both descriptions.

Bench.	***waka***		***maha***		***kim***		***jian***	
Resources	cmp: 1 +: 1 -: 1	cmp: 1 ALU: 2	cmp: 0 +: 1 -: 1	cmp: 0 +: 2 -: 3	cmp: 2 +: 2 -: 1	cmp: 2 ALU: 2	cmp: 1 +: 1 -: 1	cmp: 1 +: 2 -: 1
descr. 1	7/7/4	7/7/4	5/5/4	4/4/2	6/6/6	6/6/6	4/4/4	4/4/2
descr. 2	7/7/4	7/7/4	5/5/4	4/4/2	6/6/6	6/6/6	4/4/4	4/4/2

Table 1. Insensitivity to syntactic variance

5. Conclusion

Our HCDG-based scheduling approach exploits most of the existing scheduling optimization techniques, enjoying their combined benefits. Both speculative execution and conditional resource sharing are combined in a uniform and consistent framework similarly to dynamic CV's of [3] and guards in [4], [5]. Even more, it does not suffer from effects of syntactic variance at both the mutual exclusiveness detection and scheduling levels, as CDFG, CFG based approaches. *Structural*, *behavioral* and *data-flow* exclusiveness (as classified in [8]) can be extensively detected using graph transformations and guard information. The hierarchical control representation permits to minimize the number of mutual exclusiveness tests and also develop probabilistic priority functions that account for the conditional nature of the design. Experimental results confirm the effectiveness of both the scheduling policy and the proposed priority function.

References

[1] R. A. Bergamaschi, S. Raje, I. Nair, L. Trevillyan, *"Control-Flow versus Data-Flow-Based Scheduling: Combining Both Approaches in an Adaptive Scheduling System"*, IEEE Trans. VLSI, v. 5, n. 1, p. 82-100, 1997

[2] T. Kim, J.W.S. Liu, C.L. Liu, *"A Scheduling Algorithm For Conditional Resource Sharing"*, Proc. ICCAD 91, pp. 84-87, 1991

[3] K. Wakabayashi, T. Yoshimura, *"Global Scheduling Independent of Control Dependencies Based on Condition Vectors"*, Proc. 29th DAC, 1992

[4] I. Radivojevic, F. Brewer, *"Analysis of Conditional Resource Sharing Using a Guard-based Control Representation"*, Proc. of ICCD'95, pp. 434-439, Oct. 1995

[5] I. Radivojevic, F. Brewer, *"Incorporating Speculative Execution in Exact Control-Dependent Scheduling"*, Proc. 31st DAC, pp. 479-484, Jun. 1994

[6] R. Camposano, *"Path-based Scheduling for Synthesis"*, IEEE Trans. on CAD, vol. 10, no. 1, pp. 85-93, 1991

[7] K. Wakabayashi, T. Yoshimura, *"A Resource Sharing and Control Synthesis Method for Conditional Branches"*, Proc. of IEEE ICCAD-89, pp. 62-65, 1989

[8] J. Li, R. K. Gupta, *"An Algorithm To Determine Mutually Exclusive Operations In Behavioral Descriptions"*, Euro-DAC'97

[9] V. Chaiyakul, D.D. Gajski, L. Ramachandran, *"Minimizing Syntactic Variance with Assignment Decision Diagrams"*, UCI, Tech. Rep., ICS-TR-92-34, Apr. 1992

[10] Y-L. Lin, *"Recent Developments in High-Level Synthesis"*, ACM Trans. on Design Automation of Electronic Systems (TODAES), v. 2, n. 1, Jan. 1997, pp. 2-21

[11] T. Kim, N. Yonezawa, J.W.S. Liu, C.L. Liu, *"A Scheduling Algorithm For Conditional Resource Sharing - A Hierarchical Reduction Approach"*, IEEE Trans. on CAD, vol. 13, no.4, pp. 425-438, Apr. 1994

[12] A.C. Parker, J.T. Pizarro, M. Mliner, *"MAHA: A Program for Data Path Synthesis"*, Proc. 23rd DAC, pp. 252-258, 1986

[13] R. Puri, J. Gu, *"An Efficient Algorithm for Microword Length Minimization"*, Proc. DAC'92, pp. 651-656

[14] C.J. Tseng, D.P. Siewiorek, *"Automated Synthesis of Data Paths on Digital Systems"*, IEEE Trans. on CAD, vol. 5, no.3, pp. 379-395, Jul. 1986

[15] R. A. Bergamaschi, R. Camposano, M. Payer, *"Allocation Algorithms Based on Path Analysis"*, Integration, The VLSI Journal, vol.13, no.3, p. 283-99, Sept. 1992

[16] L.C.V. dos Santos, J.T.J van Eijndhoven, J.A.G. Jess, *"Combining Code Motion and Scheduling"*, ProRISC '96, Mierlo, the Netherlands

[17] A. Kifli, G. Goossens, H. De Man, *"A Unified Scheduling Model for High-Level Synthesis and Code Generation"*, Proc. EDTC'95, Mar. 1995, Paris, pp. 234-238

[18] A. Kountouris, C. Wolinski, *"Hierarchical Conditional Dependency Graphs for Conditional Resource Sharing"*, Proc. Euromicro98, Wasteras, Sweden

[19] A. Kountouris, C. Wolinski, *"Extensive Conditional Resource Sharing Based on Hierarchical Conditional Dependency Graphs"*, to appear in Proc. 12th Int'l VLSI Conference, Goa, India, Jan. 1999

[20] L. Besnard, *"Compilation de SIGNAL: Horloges, Dependances, Environment"*, Ph.D., Univ. of Rennes I

[21] T. P. Amagbegnon, *"Forme Canonique Arborescente des Horloges de SIGNAL"*, Ph.D., Univ. of Rennes I, Dec. 1995