A Slice-Based Automatic Hardware/Software Partitioning Heuristic

H. Parandeh-Afshar, A. Tootoonchian, M. Yousefpour, O. Fatemi, and M. Hashemi

Nanoelectronics Center of Excellence, School of Electrical and Computer Engineering,
University of Tehran, Tehran, Iran

**hparande@ut.ac.ir, a.tootoonchian@parsemi.com, m.yousefpour@ece.ut.ac.ir,
omid@fatemi.net, hashemi@comnete.com**

***Abstract*— In this paper, a novel level-based hardware/ software partitioning heuristic has been proposed. The algorithm operates on functional blocks of designs represented as directed acyclic graphs (DAG), with the objective of minimizing the processing time under various hardware area constraints. In most existing methods, the communication overhead and the fact that the vertices mapped onto the same computing unit have less communication, is overlooked during the partitioning decision, while the proposed algorithm considers this fact during partitioning.**

I. INTRODUCTION

Hardware-Software codesign transforms an application specification into interacting hardware (HW) and software (SW) components that exhibit the desired behavior and satisfy the performance constraints. Task partitioning and task scheduling are required in many applications. Sub-tasks extracted from the input description should be implemented in the right place (using the partitioner) at the right time (using the scheduler). Optimization techniques based on *heuristic methods* are generally employed to explore the search space so that feasible and near-optimal solutions can be obtained. Common heuristic-based algorithms include tabu search (TS), simulated annealing (SA), and genetic algorithm (GA). These algorithms mostly are application independent and do not use the specific characteristics of different applications. For instance, a data flow application has different characteristics from a control bound application. In the first set of applications, the application is specified by the flow of data while the later is inherently based on control signals. Using the characteristic of data flow applications a new heuristic algorithm has been proposed in this paper. Generally, in this paper, the problem of minimizing execution time of an application for a system with hard area constraints is considered. It is assumed that an application is specified as tasks graph DAG (directed acyclic graph) extracted from a sequential application written in *C* or any other procedural language, where the graph vertices represent functions, and the graph edges represent function calls or accesses between functions.

In this paper, we propose a level-based HW-SW partitioning heuristic which is applied to data flow systems. In this heuristic, a limited search space is used for performing partitioning. This space is named *window*. In most existing methods, the communication overhead and the fact that the vertices mapped onto the same computing unit have less communication, is overlooked during the partitioning decision, while the proposed algorithm considers this fact during partitioning. Also it will be shown how with an adaptive cost function, area constraint is applied. Simulation results show improvements in search time while keeping the same quality of solution when compared to three other heuristic search algorithms: genetic algorithm (GA), simulated annealing (SA) and tabu search (TS).

The rest of this paper is organized as follows: In Section 2, a review of related works in HW-SW partitioning is provided, while in Section 3, problem description will be defined and the reference architecture that has been used is introduced. Section 4 discusses the proposed partitioning algorithm. Experimental results and comparisons will be presented in Section 5. Finally, the summary and concluding remarks are given in Section 6.

II. RELATED WORK

Partitioning is a crucial step in codesign because it plays an important role in allocating tasks properly between hardware and software under system constraints. Several approaches such as dynamic programming [2], genetic algorithms [3], and greedy heuristics [4] have been proposed. Most of the initial work, [2], [5], focused on the problem of meeting timing constraints with a secondary goal of minimizing the amount of hardware. Subsequently there has been a significant amount of work on optimizing performance under area constraints, [6], [7], [8]. With the goal of searching a larger design space, techniques such as

1-4244-0765-6/06/$20.00 ©2006 IEEE

simulated annealing (SA) have been applied to HW-SW partitioning using fairly simple cost functions. While a lot of initial work such as [5] was based exclusively on SA, recent approaches commonly measure their quality against a SA implementation. For example, [6] compares simulated annealing with a knowledge-based approach, and [7] compares SA with Tabu search.

The three previous works in HW-SW partitioning that are most directly related to our work are [7], [8], [10]. The reference model of our work for HW-SW partitioning is based on [9], in which partitioning granularity is at task level: each partitioning object represents a function and the DAG edges are annotated with call counts. Similar to [7], [8], and [10], the problem of minimizing execution time is considered while satisfying HW area constraints.

III. System Architecture

In this paper the problem of HW-SW partitioning of an application specified as a task graph extracted from a sequential program written in *C* is considered. For the purpose of illustrating the basic partitioning formulation, we assume a simple target architecture that contains one SW processor and one HW unit connected by a system bus. Both software and hardware have their own local memory and communicate with each other through a shared memory (system memory) and shared resource conflicts are taken into account. Waiting time is added when a shared resource is engaged by another task, and a waiting task can only be executed after the shared resource is released (non-preemptive).

The input to the partitioning algorithm is a DAG representing a task-graph, TG = (V, E). *V* is the set of graph vertices and *E* is the set of edges. Each partitioning object corresponding to a vertex $vi \in V$ is essentially a function that can be mapped to HW or SW. Each edge $eij \in E$ represents a call or an access to the callee function *vj* from the caller function *vi*. The SW execution times and call counts are obtained from profiling the application on the SW processor. The simple model for HW area estimates assumes that the area of a cluster of components can be obtained by summing the individual HW areas. Communication time estimates are made by simply dividing the volume of data transferred by the bus speed.

Here as [7]'s approach, heuristic algorithm is used as the partitioner. The partitioning results will be evaluated by a cost estimation model to obtain processing time. Estimating processing time is based on list scheduling, in which hardware tasks are scheduled without resource conflicts.

The processing time includes communication time, execution time on hardware or software, and waiting time for available shared resources or input data. Task allocations are obtained as a result of scheduling, and no resource conflict is guaranteed. Such an approach is applicable to coarse grain or *functional* partitioning and scheduling.

The list scheduler is used to order tasks, without any shared resource conflicts, with regard to partitioning results, task precedence, and the target system model. After the processing time information is obtained, it is sent back to guide the partitioner to explore only promising regions. This iteration process between partitioning and scheduling to minimize processing time will terminate when the stop condition (such as the number of iterations specified by the designer) is met.

IV. Partitioning Heuristic

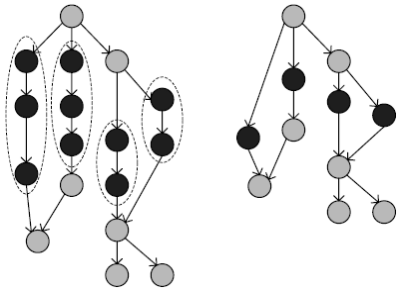
Proposed partitioning algorithm consists of two main steps: *Clustering* and *Windowing*. These two steps are discussed in the following.

A. Clustering Algorithm

Due to the communication costs and limitations, we propose a clustering algorithm to merge nodes in the graph. By merging of nodes, more coarse grained nodes are constructed and dedicated communication links are established between nodes in a cluster.

The other advantage of clustering is reducing the search space which affects partitioning search time. In this work, the nodes that are closely related are merged. Closely related nodes are the nodes which in simple chains of the graph. A simple chain contains the nodes that have only one successor and one predecessor. The nodes that make a simple chain are merged together where the merging process continues until the overall process time of the cluster, when it is implemented in HW, is below a threshold. This constraint prevents from making very coarse grained clusters in which smaller nodes could have better partitioning. The threshold is two times greater than the maximum delay of nodes in the graph when they are implemented by HW.

Figure 1 (a) shows a DAG which has several simple chains. In this figure, four sets of simple chains have been shown. The nodes in each chain can be merged by considering the threshold. The new clustered graph is shown in Figure 1 (b). In this new graph there are fewer nodes than the previous one. This graph will be used for partitioning by applying the proposed partitioning heuristic.



(a) Original DAG (b) Clustered DAG

Fig. 1. Node Clustering of DAG

B. Partitioning Heuristic

First, the clustered graph of previous section is used for constructing a precedence graph. In this step, each node in

the graph is assigned a level. This new graph is named *precedence* graph. In the precedence graph, the nodes that have the same level are independent and hence, can be executed concurrently. Therefore, these concurrent tasks can be extracted from the precedence graph. This work is performed by using the data dependency of tasks in the data flow graph. In this stage, a level is assigned to each node which is used in the proposed partitioning process.

In the precedence graph, concurrent tasks have been defined and they could be executed simultaneously. On the other hand, the successor nodes should wait until their predecessor nodes finish their execution. This means that in the partitioning process, an incremental algorithm for defining HW or SW assignment can be used by considering the levels of the tasks in the graph. Using this fact, the search space of partitioning process can be limited. In the simplest case, the nodes in each level can be selected for partitioning. Limiting search space to members of a level instead of all of the system will decrease complexity of the large problems to few nodes of each level.

On the one hand, limiting the process of partitioning decision to single levels can not produce efficient results and some more levels should be considered in each iteration. On the other hand, using more levels for making decision will increase search time. To solve this problem, the overall search space is divided to some slices. These slices are named *window*. A window is a space which consists of several consequent levels. In each step, it is tried to optimize the current window search space. So an exhaustive search is performed to find the most optimum result. The most optimum result has the minimum cost in terms of processing time of each task and communication overhead between tasks. Meanwhile, the exact communication overhead is specified after scheduling. Hence, for evaluating the result of each window, partitioning results are sent to the list scheduler and the overall process time of that window is computed. This window moves along the levels of the graph until it reaches the end of the graph that is when partitioning process ends.

As mentioned in the problem description section, the proposed system architecture covers important features of the target system such as shared resource conflicts and communication time overhead. In the general model, the system architecture consists of one processor (software representative) and a set of dedicated hardware. Both software and hardware have their own local memory and communicate with each other through shared memory. But the fact is that the vertices mapped onto the same computing unit, have less communication latency and the delay overhead of data communication between different systems varies. In general, hardware-hardware or software-software communication methods mostly are faster than hardware-software communication methods which are used in hardware/software co-design [11] [12]. On the other hand, one of the most important issues that should be considered in codesign process is communication overhead. This issue is especially important during partitioning process where different tasks are assigned either to HW or SW.

As a result, although the process time of each task is an effective issue in making decision about HW or SW assignment, the mapping of parent nodes should be considered, too. Therefore, in the proposed algorithm, one of the factors that affect the partitioning decision is how the predecessors of each node have been partitioned. For instance, if the parents of one node are assigned to HW (SW), there would be more probability that the node to be assigned to HW (SW) with a specified weight.

In the windowing algorithm there are two issues regarding the size of the window and the way that the window moves. Experimental results showed that by setting the window size to 3, optimum results are achieved in terms of processing time and search time. The second issue is the way that the window moves across graph levels. The movement of window across the graph can be performed in two ways. The first one is moving the window such that no overlapping occurs between two consecutive windows. This model will be inefficient whereas in the proposed algorithm mapping of parent nodes affects the mapping of their successors. The second method proposes to keep overlapping between consecutive windows. In this work, one level is shared between two consecutive windows. This helps the next windows keep track of the previous mappings. Figure 2 shows a simple precedence graph with a window size of 3. As illustrated, the nodes in the last level of current window have dark color. These nodes will be used in the partitioning decision of the next window and their mappings may be changed.

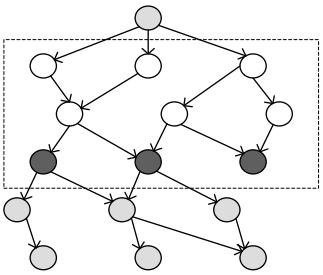


Fig. 2. A window with size = 3 and overlapped nodes.

Another issue that should be considered during the partitioning process is area constraint. The problem considered in this paper is to partition the application into HW and SW components such that the execution time of the application is minimized while simultaneously satisfying the hard area constraints of the HW unit. For this purpose, the cost function contains an area factor. This factor initially has a tentative value. In each iteration of the partitioning algorithm, area consumption will be evaluated. If the resulted partitioning meets area constraint requirements, the algorithm will be terminated; otherwise the area factor of cost function will be adaptively updated. Logically this factor should be decreased to reduce the probability of HW

assignment. This may resulting in having the nodes that have similar or close costs either in HW or SW to be moved to SW. The algorithm will iterate until the area constraint is met.

For decreasing the area factor of cost function a varied value is used. This value adaptively varies during partitioning iterations. Consider the 'α' value in the following equation.

$$\alpha = (\text{aggregate area}) / (\text{area constraint}) \qquad (1)$$

In this equation α is the result of division of aggregate area of all HW components of current partitioning by the area constraint value. If the aggregate area of all HW components is greater than area constraint, 'α' will be greater than 1. In this case, if the area factor of cost function is divided by α, it will be decreased. This process will be repeated until α reaches a value less than 1.

V. Experimental Results

For evaluating the proposed algorithm, three other heuristic algorithms, GA, SA, and TS were used for comparison.
Figure 3 shows results of these heuristics for a randomly generated task graph under various area constraints. As shown, the process time of the proposed algorithm is very close to the other three heuristic methods while the search time is about 4 times faster on average.

In this experiment, rather than using random graphs, we used graph structures that are common in real applications. These structures include out-tree, in-tree, fork-joint, mean value and FFT with a constant number of nodes and edges as shown in Table I [1]. For each graph type, 20 runs are performed. These results have been computed with the area constraint of 0.9. Results obtained from all realistic graphs clearly show that the proposed heuristic is superior to GA, SA and TS in terms of search time while keeping the same processing time.

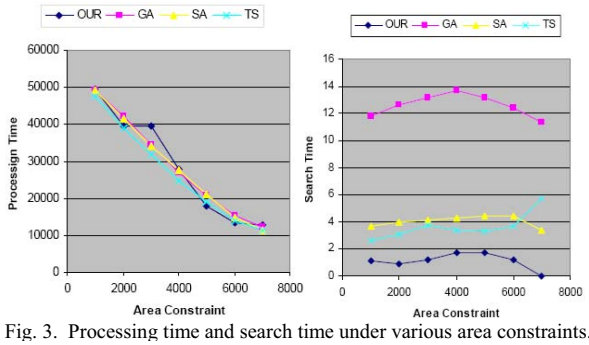


Fig. 3. Processing time and search time under various area constraints.

TABLE I
Results of some realistic task graphs

		In-Tree	Out-Tree	Fork-Joint	Mean-Value	FFT
GA	P Time	5653	5843	10698	12243	7871
	S Time	2.0	1.79	3.28	5.67	2.56
SA	P Time	5567	5823	19823	12207	7676
	S Time	1.38	1.33	2.25	3.14	1.86
TS	P Time	5599	5737	10490	12192	7911
	S Time	1.87	1.99	3.1	4.80	2.68
OUR	P Time	5353	5878	10658	11712	7798
	S Time	0.47	0.54	0.968	1.015	0.393

VI. Conclusion

In this paper, a new heuristic partitioning algorithm suitable for data flow applications was presented. In this method, the fact that the vertices mapped onto the same computing unit have less communication overhead is considered. Furthermore a novel windowing algorithm that reduces the search space and inherently improves the search time has been introduced. In this algorithm, by considering the data dependency of tasks a level is assigned to them. It was shown that by using the level of each task and constraining the search space to a window space, search time was reduced comparing to three popular heuristics, GA, SA, and TS. Experimental results indicate that the processing time of the partitioned system using the proposed algorithm is comparable to the existing techniques.

References

[1] Y.-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocation Task Graphs to Multiprocessor," *IEEE Transaction on Parallel and Distributed Systems*, vol. 7, pp. 506–521, 1996.

[2] R. Ernsl, J Hcnkel, and T Benner, "Hardware-software cosynthesis for microcontrollers". IEEE Design and Test,V-10, Dec 1993.

[3] K. Ben Chehida, M. Auguin, "HW/SW partitioning approach for reconfigurable System design", CASES 2002.

[4] A. Kalavade, E. Lee, "A global criticality/Loca1 Phase Driven algorithm for the Constrained Hardware/Software partitioning problem", CODES 1994.

[5] R. Gupta, De. Micheli, "System-level synthesis using re-programmable components", EDAC 92.

[6] M. L. Vallejo, J. C. Lopez, "On the hardware-software partitioning problem: System Modeling and partitioning techniques", ACM TODAES, V-8, 2003.

[7] T. Wiangtong, P.Y.K. Cheung, W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in HW-SW Codesign", *International Journal on Design Automation for Embedded Systems*, vol. 6, 2002.

[8] F Vahid, TD Le. "Extending the Kernighan-Lin heuristic for Hardware and Software functional partitioning", Jrnl Design Automation for Embedded Systems, V-2, 1997.

[9] P. Eles, Z. Peng, K. Kuchinski, Doboli, "System Level Hardware/Software Partitioning based on simulated annealing and Tabu Search", Jrnl Design Automation for Embedded Systems, V-2,. 1997.

[10] J. Henkel, R. Ernst, "An approach to automated hardware-software partitioning using a flexible granularity that is driven by high-level estimation Techniques". IEEE Trans. On VLSI, V-9 2001.

[11] Peter Voigt Knudsen et al, "Integrating Communication Protocol Selection with Partitioning in Hardware/Software Codesign", Proceedings of the 11th international symposium on System synthesis, 1998.