A Path Analysis based Partitioning for Time Constrained Embedded Systems

Luc Bianco, Michel Auguin, Guy Gogniat, Alain Pegatoquet
I3S - Université de Nice Sophia Antipolis - CNRS
41 Bd Napoléon III - 06041 Nice cedex
email: *lastname*@alto.unice.fr

Abstract

The HW/SW partitioning problem addressed in this paper is one of the key steps in the co-design flow of heterogeneous embedded systems. Generally the aim is to provide solutions that respect timing constraints and minimize an objective function such as the total area and/or the power consumption. Minimizing the hardware area conflicts with reducing execution time. Therefore, we introduce an heuristic for synthesizing heterogeneous systems that uses a global metric to guide the mapping of tasks according to the reusability of components and the time margin induced by timing constraints.

1. Introduction

With advances of ASIC technologies and strong market pressures there is a major interest in the investigation of efficient co-design methodologies for embedded system design. These systems are subject to stringent real time and embedding constraints such as silicon area and/or power consumption. Furthermore, due to the different nature of processing involved in application specific systems, an heterogeneous architecture model composed of multiple software components (e.g., DSPs and RISCs) coupled with dedicated hardware units is generally the only practical way to fit application requirements. Therefore, numerous works have focused on hardware/software partitioning which is a key step in the co-design flow. For example, approaches in [6],[8],[10] target monoprocessor based systems. Mutual exclusion of hardware and software parts is assumed in [7]. In [3] the aim is to determine a distributed real time system composed of processing elements that may perform preemptive schedule. In [12] is presented an extension of the Kernighan/Lin algorithm for functional partitioning but it assumes that the set of units of the target architecture is defined first.

Our aim is to provide solutions that respect real time constraints and minimize the silicon area. Minimizing the area implies generally to make the best reuse of resources. Since this objective may conflict with timing constraints, it is of prime importance to dispose of a global metric able to guide the partitioning heuristic in the allocation of components. In [8], a global criticality metric is based on the assumption that the target system contains only one SW component, exploiting then the sequential nature of software processing. With heterogeneous multiprocessor systems, this assumption is not valid yet and a task may have several SW implementations [9]. Consequently, we use a path length metric based on the critical path length introduced in [1]. But the aim of [1] is to achieve the fastest schedule of a set of tasks on a predesigned system defined as a set of interconnected processors.

2. Hardware-software co-design

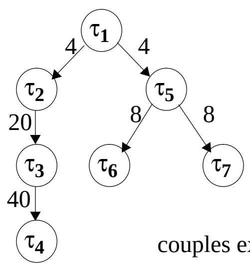
2.1. The target architecture

Since the complexity of applications and ASIC technologies are both increasing continuously, it is of prime importance to deal with systems on a chip composed of several processor cores associated with dedicated functional units. Furthermore, according to the nature of processing involved in the application, different types of processor cores (e.g., RISC and DSP) could be integrated in the target architecture. Therefore, based on a library of HW and SW components, each one executing one or several tasks, the aim of partitioning is to select the set of components that corresponds to the better implementation of the application, i.e. the implementation which respects timing constraints and minimize an objective function. In the sequel, our partitioning algorithm attempts to minimize the total area of the system but other objective functions could be investigated.

2.2. System specification

Numerous static signal processing applications can be described with an acyclic data flow graph (DFG) [2],[11], $G_T=(V_T,E_T)$, where nodes ($\tau_i \in V_T$) represent tasks, and edges ($e_{ij} \in E_T$) represent dependencies. Each edge is annotated with the volume of data $d_{i,j}$ to be transferred between τ_i and τ_j. For sake of simplicity a single timing constraint T_{max} is set up for all the ending nodes of G_T but extending the algorithm with different timing constraints is straightforward. In figure 1 is given an example (issued from [1]) of a DFG representation of a system.

The function area is the incremental area that must be added to the system to implement the function. Each task must have at least one realization model in the library of HW and SW units. In the example depicted in figure 1 each task has as many realization models than units in the library.



Task	Type	P1	P2	P3
τ_1	1	2/2	1/3	2/2
τ_2	2	4/10	1/15	3/12
τ_3	3	6/1	2/5	3/4
τ_4	4	5/3	3/8	4/4
τ_5	5	3/3	3/3	2/6
τ_6	3	6/1	2/5	3/4
τ_7	6	3/2	1/4	1/6

couples execution time/ function area

Fig. 1 • An acyclic Data Flow Graph

3. Path Analysis based Partitioning Algorithm (PA^2)

One of the major problems encountered by partitioning heuristics is to evaluate at each decision step the impact implied by mapping a task on a component on future iterations of the algorithm [9]. We use an extension of the path analysis introduced in [1] to evaluate the time criticality resulting of the assignment of a task to a component.

3.1. Communications between components

The partitioning algorithm uses a simple communication model to evaluate delays due to data transfers between tasks:

$$tc_{k,m}(\tau_i,\tau_j) = \frac{d_{i,j}}{D_{k,m}} \quad \text{if } k \neq m\text{, 0 if } k = m.$$

where $d_{i,j}$ is the number of data to be transferred between tasks τ_i et τ_j on units k and m and $D_{k,m}$ is equal to the lowest input and output throughputs of units k and m.

3.2. Path length evaluation

PA^2 is based on the study of the path lengths issued from a task to any ending task of the graph. The approach presented in [1] attempts to determine the partition that minimizes the total execution time and only minimum critical path lengths are evaluated. Since the aim of PA^2 is to provide solutions that respect timing constraints and minimize the total area, we consider both minimum and maximum path lengths. The minimal (resp. maximum) path length pl_{min} (resp. pl_{max}) from a task τ_i is the shortest (longest) path in the DFG from τ_i to any ending task. The evaluation of these paths takes data communications into account:

$$pl_{min}(\tau_i,u_k) = t_{exe}(\tau_i,u_k) + Max\left(Min_{\tau_j \in succ(\tau_i),u_l}\left(tc_{u_k,u_l}(\tau_i,\tau_j) + pl_{min}(\tau_j,u_l)\right)\right)$$

$$pl_{max}(\tau_i,u_k) = t_{exe}(\tau_i,u_k) + Max_{\tau_j \in succ(\tau_i),u_l}\left(tc_{u_k,u_l}(\tau_i,\tau_j) + pl_{max}(\tau_j,u_l)\right)$$

where $t_{exe}(\tau_i,u_k)$ is the execution time of the task τ_i on u_k, $succ(\tau_i)$ represents the set of direct successors of task τ_i and u_l is the component that executes τ_j. These two path lengths are evaluated for each task and each component of the library. The longest path length $pl_{max}(\tau_i,u_k)$ whatever τ_i, a source node of the DFG, is noted PL_{MAX}. An example of computation of the minimum path issued from τ_5 (figure 1) is given in figure 2.

Task execution time Execution times of successors

$$pl_{min}(\tau_5,P1) = 3 + Max(Min(0+6,2+2),Min(0+3,2+1)) = 7$$

Task Component Communication times

Fig. 2 • Example of a path length computation

From a given task, the path lengths provide the estimated boundaries on time delays until complete executions of ending tasks of the graph. Evaluations of these paths do not depend on the schedule of tasks and can be performed in a former static step of the partitioning algorithm [1].

The partitioning heuristic is based on a scheduling algorithm that determines an assignment and a schedule to data ready tasks, i.e. tasks whose predecessors have been scheduled. In order to guide the partitioning process, the algorithm computes available times associated with couples <task, implementation model>. The computation of an available time takes into account the time constraints settled on ending tasks, path lengths issued from the task and instants of availability of data deduced from the schedule of predecessors of the task.

3.3. Available times of tasks

For each data ready task, available times are computed for any HW and SW components that are able to realize it. Since the partitioning algorithm is based on an iterative process, we distinguish instances of components that have been added in the architecture in previous iterations from new instances deduced from models of the component library. The available time $T_a(\tau_i,u_k)$ of a task τ_i associated to a new instance u_k is given by:

$$T_a(\tau_i, u_k) = T_{max} - Max_{\tau_j \in pred(\tau_i)}\left(t_{end}(\tau_j) + tc_{p(\tau_j), u_k}(\tau_j, \tau_i)\right)$$

where $p(\tau_j)$ is the component in the architecture executing τ_j. This time depends only on availability of data. The available time $T_a(\tau_i,u_k)$ associated with an instance u_k of the architecture depends also on the utilization of u_k by tasks that have been previously scheduled on it:

$$T_a(\tau_i, u_k) = T_{max} - Max\left(Max_{\tau_j|_{p(\tau_j) = u_k}} t_{end}(\tau_j),\ Max_{\tau_j \in pred(\tau_i)}\left(t_{end}(\tau_i) + tc_{p(\tau_j), u_k}(\tau_j, \tau_i)\right)\right)$$

In practice, an accurate value of $T_a(\tau_i,u_k)$ is evaluated by scanning the idle time slots of u_k in order to find the earliest time slot greater than $t_{exec}(\tau_i,u_k)$ and such that data required by τ_i are available.

Two relations between path lengths and task available times are obvious (Figure 3):

- (a) If $pl_{min}(\tau_i,u_k) < T_a(\tau_i,u_k)$ and $pl_{max}(\tau_i,u_k) < T_a(\tau_i,u_k)$ then scheduling τ_i on u_k is not harmful for successive partitioning iterations.
- (b) If $pl_{min}(\tau_i,u_k) > T_a(\tau_i,u_k)$ and $pl_{max}(\tau_i,u_k) > T_a(\tau_i,u_k)$ then the schedule of τ_i on u_k leads to the obvious failure of the partitioning due to timing constraint violations.

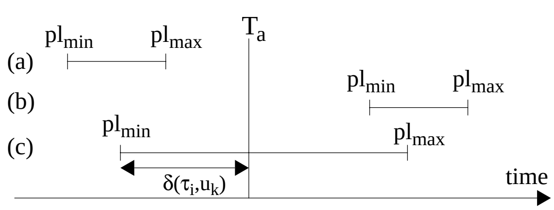


Fig. 3 • Task allocation times

But these relations are not sufficient to guide the partitioning algorithm in all cases (Figure 3.c). Other metrics are introduced to help the assignation process.

- Relative execution time drop: $e(\tau_i,u_k)$

This value represents the difference between the execution time of τ_i by u_k and the lowest execution time of τ_i by components of the library:

$$e(\tau_i, u_k) = t_{exe}(\tau_i, u_k) - Min_{u_l \in library}(t_{exe}(\tau_i), u_l)$$

- Maximum freedom of schedule: $\delta(\tau_i,u_k)$

This value gives the maximum available time frame to schedule τ_i (and successors of τ_i) on u_k (Figure 3):

$$\delta(\tau_i, u_k) = T_a(\tau_i, u_k) - pl_{min}(\tau_i, u_k)$$

Components with the largest values of $\delta(\tau_i,u_k)$ are those that impose the lowest constraints on the partitioning.

- Discrimination factor: $\gamma(\tau_i,u_k)$

$$\gamma(\tau_i, u_k) = 1 - \frac{e(\tau_i, u_k)}{\delta(\tau_i, u_k)} \times \frac{pl_{max}(\tau_i, u_k)}{PL_{MAX}}$$

The discrimination factor is a weighted relative value giving the amount of time taken by u_k on the available time frame $\delta(\tau_i,u_k)$ associated with τ_i and its successors. This factor is weighted by the relative task position to lessen its influence as the algorithm proceeds.

3.4. Task partitioning

Based on this set of metrics, the partitioning algorithm iterates with data ready tasks. Let τ_i and A_n be respectively a data ready task and an architecture under construction.

In a first phase, the algorithm encourages the reuse of instances of A_n. With instances $u_k \in A_n$ able to realize τ_i the algorithm constructs three lists (Figure 4):

- List L^1_{reuse}: In this list are placed instances $u_k \in A_n$ such that $pl_{max}(\tau_i,u_k) \le T_a(\tau_i,u_k)$.
- List L^2_{reuse}: This list contains instances $u_k \in A_n$ such that $\gamma(\tau_i,u_k) \ge T_{reuse}$ where T_{reuse} is a threshold value set up by the designer. Setting T_{reuse} close to 0 enforces the reuse of instances. On the contrary, fixing T_{reuse} close to 1 attempts to provide solutions with lower execution times since the algorithm keeps a greater temporal margin according to time constraints.
- List L_{others}: Remaining instances of A_n able to realize τ_i and such that $pl_{min}(\tau_i,u_k) \le T_a(\tau_i,u_k)$ are placed in the list L_{others}.

With new instances derived from the library, the algorithm constructs a new list L_{new} and updates the list L_{others}:

- List L_{new}: This list contains new instances u_k such that $\gamma(\tau_i,u_k) \ge T_{new}$ where T_{new} is a second threshold value set up by the designer. If T_{new} is close to 1 the algorithm attempts to provide solutions with fast components. If T_{new} is close to 0 the algorithm tries to minimize the total silicon area.

• List L_{others}: This list is updated with new instances u_k such that $pl_{min}(\tau_i,u_k) \leq T_a(\tau_i,u_k)$.

The partitioning algorithm operates as follows:

while all the tasks of G_T are not scheduled **do**
select data ready tasks τ_i;
foreach implementation model u_k of τ_i **do**
compute $\gamma(\tau_i,u_k)$; place u_k in the corresponding list;
endfor;
caseof
$L^1_{reuse} \neq \emptyset$: select $u_k \in L^1_{reuse}/pl_{max}(\tau_i,u_k)$ minimum; **break**;
$L^2_{reuse} \neq \emptyset$: select $u_k \in L^2_{reuse} / \gamma(\tau_i,u_k)$ maximum; **break**;
$L_{new} \neq \emptyset$: select $u_k \in L_{new}$ / Area(u_k) minimum; **break**;
$L_{others} \neq \emptyset$: select $u_k \in L_{others}$ / **if** $T_{new}+T_{reuse}>1$ then $\gamma(\tau_i,u_k)$ maximum else Area(u_k) minimum **endif**; **break**;
otherwise: error("the partitioning is not feasible"); **break**;
endcase;
schedule as soon as τ_i on u_k according to available time slots of u_k;
endwhile;

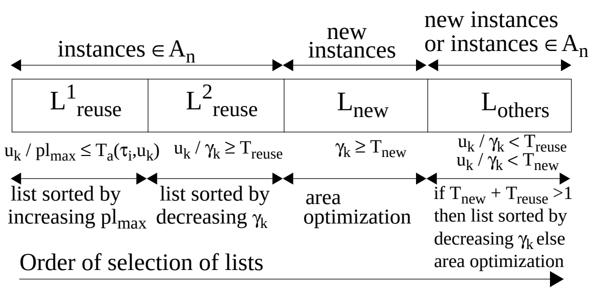


Fig. 4 • The partitioning task

Note that since the effect of the discrimination factor $\gamma(\tau_i,u_k)$ is lessened as the algorithm proceeds, the number of instances placed in lists L^2_{reuse} and L_{new} increases and in the meanwhile the temporal margin kept by the algorithm decreases. This feature permits to approach the time constraints when ending tasks and tasks close to ending tasks are scheduled.

4. Results

This algorithm has been developed in C. Results about the simple example given in figure 1 are detailed first. We consider three different time constraints: 9,14 and 20 time units. Values of minimum and maximum path lengths are given in figure 5. Results of our partitioning algorithm and a comparaison with the HCP algorithm[1] are illustrated in figure 6. If the time constraint is set up to 9 time units, PA^2 with T_{reuse}=0.2 and T_{new}=0.9 provides the same results than HCP, i.e, 8 time units and two components (P2 and P3). When T_{max}=14, PA^2 approaches the time constraint(13 time units) with only one component (P2) and a total area of 43 units. When T_{max}=20, PA^2 selects only the component P3 giving an execution time of 18 time units and a total area of 38 units.

task	$pl_{min}(\tau_i,P1)$	$pl_{min}(\tau_i,P2)$	$pl_{min}(\tau_i,P3)$	$pl_{max}(\tau_i,P1)$	$pl_{max}(\tau_i,P2)$	$pl_{max}(\tau_i,P3)$
τ1	9	7	9	31	30	30
τ2	14	6	10	27	26	28
τ3	11	5	7	20	17	18
τ4	5	3	4	5	3	4
τ5	7	5	5	9	11	10
τ6	6	2	3	6	2	3
τ7	3	1	1	3	1	1

Fig. 5 • Results of the static phase

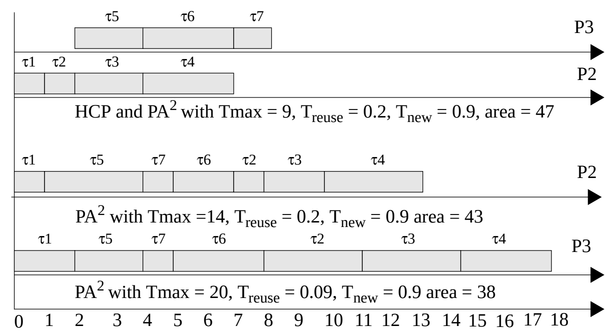


Fig. 6 • Partitioning and scheduling results

Another example corresponds to a frequency domain block adaptive acoustic echo canceller (GMDFα [4]) whose DFG is given in figure 7. We consider a DSP56002

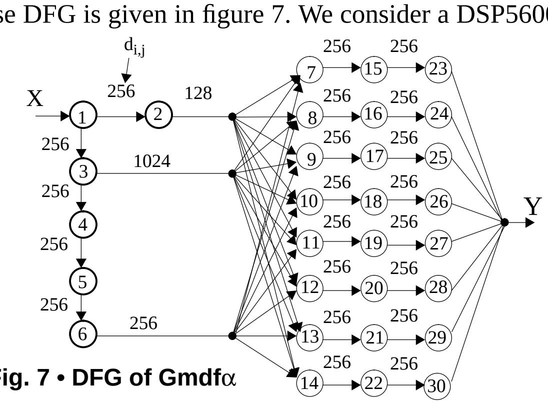


Fig. 7 • DFG of Gmdfα

processor with a mean I/O throughput of 10^7 words/second and two hardware components (figure 8). Static values used by the algorithm are also given. The sampling frequency imposes a time constraint of 8 ms. With a software only solution, there is a time constraint violation. Figure 9 shows results of partitioning that illustrate the effect of threshold values. In the second column, the algorithm assigns a software component SW to the initial task 1. This unit is used again, until the task 19 is reached where $\gamma(19,SW) < T_{reuse}$. Then a hardware unit HW2 is added and remaining tasks of the graph are scheduled on it, in order to match the time constraint. In this solution, FFT type tasks are executed without parallelism by both HW2 and SW. By increasing T_{new} to 0.96, this situation is

Tasks	Type	SW	HW1	HW2	$pl_{min}(\tau_i,sw)$	$pl_{min}(\tau_i,hw1)$	$pl_{min}(\tau_i,hw2)$
1	fft	406/ 0.15		87/1.8	1098		829
2	div	472/ 0.13			692		
3	conv/ add2d	730/ 0.21	123/ 0.86		1153	521	
4	ifft	470/ 0.16		87/1.8	806		498
5	mult/ sub	62/0.08	16/0.86		382	311	
6	fft	406/ 0.15		87/1.8	639		345
7 to 14	conv/ mult	84/0.11	23/0.86		294	208	
15 to 22	ifft	470/ 0.16		87/1.8	593		185
23 to 30	fft/add1d	451/ 0.16		98/1.87	451		98

Fig. 8 • Gmdfα: static values

Tasks	T_{reuse}=0.85, T_{new}=0.9	T_{reuse}=0.4, T_{new}=0.96	T_{reuse}=0.93, T_{new}=0.96
1	SW	HW2	HW2
2	SW	SW	SW
3	SW	SW	HW1
4	SW	HW2	HW2
5	SW	SW	HW1
6	SW	HW2	HW2
7 up to 14	SW	SW	HW1
15 up to 18	SW	HW2	HW2
19 up to 22	HW2	HW2	HW2
23 up to 26	SW	HW2	HW2
27 up to 30	HW2	HW2	HW2
Total area[a]	S = 2.56 mm^2	S = 2.4 mm^2	S = 2.86 mm^2
Execution time	T = 7.41 ms	T = 3.23 ms	T = 2.12 ms

Fig. 9 • Gmdfα: partitioning results

a. Area values neither include data memories nor hardware resources for data transfers

avoided since the algorithm selects for the task 1 a HW2 unit in place of the SW unit and maps all the successive FFT type tasks on HW2. This leads to a lower execution time with a similar area. Notice that HW1 is not used in these cases since the task 2 imposes a SW unit and for successive tasks τ_i supported by both SW and HW1, the condition $\gamma(\tau_i,SW) > T_{reuse}$ holds. If T_{reuse} is set up to 0.93 then we get solution of the third column. These results show that PA^2 permits a codesign space exploration by adjusting threshold values according to design objectives.

5. Concluding remarks and future works

Our algorithm performs the HW/SW partitioning and the scheduling of a time constrained static task graph specification and constructs a model of an heterogeneous system that may have one or more processor cores and hardware accelerators. The aim of this algorithm is to provide solutions that respect time constraints with reduced area. Two threshold parameters dealing with the reusability of previous instances and the time margin imposed by time constraints allow to adapt the partitioning objectives in order to perform a rapid evaluation of various alternatives. By selecting adequate values to these threshold parameters the designer can investigate solutions with rapid execution times or reduced area.

Some improvements are under investigation. One of them concerns the communication model that must be extended in order to deal with more realistic structure of components. For example a DSP or a RISC core may have different types of I/O ports, each one with different timing characteristics (e.g., parallel or serial ports). Therefore, an effective selection of I/O ports combined with well suited communication resources may lead to a lower area to support data transferts between components of the architecture.

6. References

[1] P. Bjørn-Jørgensen, J. Madsen. Critical path driven cosynthesis for heterogeneous target architectures. *Proc. 5th Codes/CASHE'97, 15-19, Braunschweig, March 1997.*

[2] J. Buck, S. Ha, E. Lee, D. Messerschmitt, Ptolemy: a framework for simulating and prototyping heterogeneous systems, *Int. J. Computer Simulation, vol. 4, april, 1994.*

[3] B. Dave, G. Lakshminarayana, N Jha, COSYN: hardware-software co-synthesis of embedded systems, *Proc. DAC'97, Anaheim, CA, pp 703-708, 1997*

[4] L. Freund, M. Israël, F. Rousseau, J.M. Bergé, M. Auguin, C. Belleudy, G. Gogniat. A codesign experiment in acoustic echo cancellation : Gmdfα. *ISSS'96, La Jolla California, November 1996.*

[5] D. Gajski, F. Vahid. Specification and design of embedded HW-SW systems. *IEEE journal design and test of computers, 53-67, Spring 1995.*

[6] R. Gupta, G. De Micheli, Hardware-Software Cosynthesis for Digital Systems, *IEEE J. Design and Test of Computers, sept., pp 29-41, 1993*.

[7] J. Henkel, R. Ernst, A hardware/software partitioner using a dynamically determined granularity, *Proc. DAC'97, Anaheim, CA, pp 691-696, 1997.*

[8] A. Kalavade, E. Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem, *Int. Workshop on Hardware-Software Co-Design, sept. 22-24, Grenoble, France, 1994*

[9] A. Kalavade, E. Lee. The extended partitioning problem : hardware/software mapping and implementation bin-selection. *Proceedings Int. Workshop on rapid system prototyping, 12-18 Chapel Hill, NC, June 1995.*

[10] P. Knudsen, J. Madsen. PACE : a dynamic programming algorithm for hardware/software partitioning. *Proc. Codes/CASHE'96, 85-92. 1996.*

[11] R. Lauwereins, M. Engels, M.Ade, J.A. Peperstrete. GrapeII: a system level prototyping environment for DSP applications. *IEEE Computer, 35-43, Feb. 1995.*

[12] F. Vahid. Modifying Min-cut for hardware and software functional partitioning. *Proc. 5th Codes/CASHE'97, 43-48. Braunschweig, March 1997.*