Interface Optimization During Hardware-Software Partitioning

Laurent Freund, Denis Dupont, Michel Israël
LaMI, Université D'Evry
Bld des Coquibus
91025 Evry Cedex-France
{name}@lami.univ-evry.fr

Frédéric Rousseau
ESIM
Technopole de Chateau-Gombert
13451 Marseille cedex 20, France
rousseau@ismea.imt-mrs.fr

Abstract

This paper presents an approach allowing communication optimization during the hardware-software partitioning task. Our methodology focuses on systems represented by a data flow graph whose nodes are elements of libraries. To abstract the communication constraints, we include communication nodes in this graph. Consequently, assignment and scheduling of communications and operations can be determined together by the same partitioning algorithm. During partitioning, protocol optimization and bus scheduling are realized. We illustrate with a telecommunication system example the feasibility and the usefulness of our methodology.

1. Introduction

The design of telecommunication systems, like that of other embedded systems, combines hardware (ASIC, FPGA,...) and software (running on a processor) implementations. Developing such high-performance, and low-cost, systems is often called hardware/software codesign. While hardware leads to better performances, software reduces cost and facilitates modifications. But system cost and performances also depend on the additional communication overhead. Thus, a critical part of the design is to find the best trade-off between hardware, software and communications. During this task, different objectives can be targeted: minimization of system cost, minimization of execution time, or minimization of hardware-software communications. Our work focuses on the minimization of the system cost under timing constraints that take communication overhead into account. Indeed, the cost and the execution time of communications cannot be neglected in telecommunication systems since the data can be vectors or matrices and the hardware-software interface requires buses, arbitration logic and intermediate storage.

Telecommunication systems are generally data flow systems, easily represented by a Directed Acyclic Graph (DAG) whose nodes stand for computation tasks and whose edges describe data and control precedences between nodes (communications) and where all delays are considered deterministic. In recent publications [11] [1], a partitioning algorithm developed in our laboratory has been introduced that determines the *assignment* and the *scheduling* of the node operations of the DAG. In this paper we improve on that formulation and introduce a technique for optimizing the communications during—and not after—the partitioning task. First, we justify this approach by the description of the consequences on system architecture of performing interface optimization after partitioning. Second, in view of these consequences, we extract the fundamental characteristics which can be optimized during partitioning and we define our communication model. Last, we show that this model allows to perform the *assignment* and the *scheduling* tasks for communications and operations in concert. An acoustic echo cancellation system has been designed with this methodology and is given as example. Thus, the outline of the paper is as follows: Related work is discussed in Section 2. In Section 3, we describe communication concepts, how they are modeled and the consequences on the partitioning methodology. The methodology including communication during partitioning is presented in Section 4. Results of the example are given in Section 5, followed by the conclusion.

2. Related Work

Hardware-software codesign approaches differ from one another by their target architectures and by their objectives. However, all these approaches can be grouped in three categories.

The first one includes the approaches which allow *automatic* synthesis of the interface. Thus, the interfacing of components with incompatible protocols or different numbers of data pins is described in [9]. The interfacing of a processor to several devices by I/O port allocation or by

0-8186-7895-X/97 $10.00 © 1997 IEEE

memory mapped I/O is realized in [10]. Interface synthesis is treated by refinement tasks in [5]. After the designer selects among four implementation models covering a mix of global and local memories and buses, refinement procedures automatically generate the interface details. In [6], data FIFOs and a control FIFO are proposed to interface systems including unbounded delay operations.

The second category comprises approaches for optimizing communications after partitioning. The first optimization technique seeks a better *scheduling* than the one given by the partitioning task. This rescheduling of operations allows the conversion of some communications from blocking to nonblocking [4]. Thus, the control logic for *handshaking* and the number of queues for *buffered communications* can be minimized. The second optimization technique separates the specification of the communication from the implementation [12] and chooses from a library the best implementation with respect to the specification. Thus, communication realization is cast into an *allocation* problem [3]. Algorithms choose the communication units which minimize the cost of communications and satisfy technical constraints (protocol, bandwidth,...).

The third category includes methodologies dealing with communications during the partitioning task. Some of them consider communications as a criterion for hardware-software partitioning, but do not consider the sharing of communications resources (e.g. buses). Consequently, they include a constant communication cost and timing overhead in the partitioning model [8], [7]. Bus sharing during partitioning is solved by the MILP partitioning approach [2], which schedules transfers on one bus. The scheduling and the minimization of the communication cost for system components running at different clock rates is determined during partitioning in [13].

However, in the last two approaches the communication cost is restricted to the bus cost (i.e. the number of buses) and while some methods perform optimization after partitioning, the minimization and the sharing of the memory elements (for buffered transfers) during partitioning are not treated. Thus, we propose a communication model including bus scheduling and memory elements that allows optimization during partitioning.

3. Our Communication Approach

3.1. Communication Constraints

This section describes the impact of communications on partitioning and the two main reasons for choosing to implement and schedule communications during—and not after—this task.

First, the *assignment* to hardware or software for system nodes must depend on communication delay and communication cost (cost of memory, control logic and bus). Otherwise, the gain obtained by a hardware assignment can be lost in the time to transfer data to the node. Moreover, communication cost overhead can be too high.

Second, node *scheduling* and communication protocol are interdependent. The protocols taken into consideration in our study are the *blocking* protocol, which allows synchronous transfers, and the *non-blocking, buffered* or *unbuffered*, protocol for asynchronous transfers. The blocking protocol requires control logic for handshaking and it prolongs the execution time of the sender node until the transfer is done. The buffered protocol requires a storage element (FIFO) but the sender does not wait for the receiver. The unbuffered protocol does not require control logic and storage elements but the "*write*" operation of the sender must coincide with the "*read*" operation of the receiver.

Figure 1 gives an example of the difference between communication synthesis after partitioning and communication synthesis during partitioning. This example depicts two schedulings (a.1, b.1) obtained by a partitioning algorithm which minimizes cost under a timing constraint ($Tmax$). Partitioning approaches neglecting the cost of

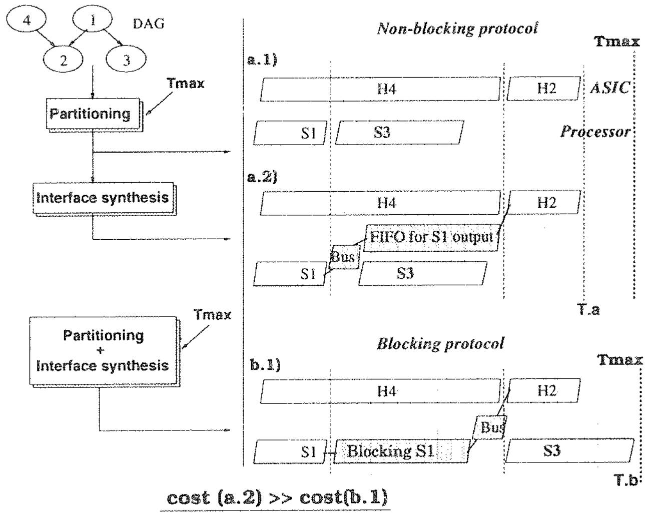

Figure 1. Scheduling and protocol

protocol during partitioning will choose solution a.1 since the execution time of the system is shorter ($T.a < T.b$). However, the resulting scheduling of node 3 imposes a non-blocking buffered protocol (FIFO) for the transfer between node 1 and node 2.

An algorithm accounting for communication cost during partitioning will have to choose solution b.1, which allows a blocking protocol without FIFO cost overhead (cost (a.2) >> cost (b.1))

Like the interdependence between scheduling and communication protocol, there also exists an interdependence between scheduling and transfer unit. We distinguish a transfer directly addressed by the processor from a transfer addressed by a DMA co-processor. CPU transfer is generally faster than DMA (because of the channel initialization) but DMA releases the processor during the transfer. Thus, a good scheduling and a good choice of communication unit allow to optimize the execution time of the system.

These two reasons show the necessity of accounting for communications during the partitioning task. Consequently, our work focuses on the possibility of modifying our methodology in order to assign and schedule nodes relating to communication constraints, and to determine protocols and transfer units during partitioning.

3.2. Communication Model for Partitioning

According to Section 3.1, a model should capture the following properties to be well adapted to partitioning:
• Communications have a cost: storage elements (FIFO), bus and synchronization logic cost.
• Communications are not immediate. Their execution time depends on volume of data, protocol and transfer unit.
• Communications call for resource sharing. For example, a transfer realized by the processor unit requires the processor resource. In the same way, buses and FIFOs are sharable resources.

In the DAG, a cost, an execution time and a set of resources are associated to each node corresponding to an operation. These standard properties are precisely those appearing in our characterization of communications. Consequently, we decided to use the same model for the communications as for the functional units of the system: nodes in the DAG. Thus, transfer is represented by a communication node set between two operation nodes.

As for other nodes, each implementation of a communication node consists in a sequence of used resources. Thus, we have defined a generic model that consists in three phases: blocking sender or data memorization into a FIFO, transfering on the bus (CPU / DMA), and blocking the receiver. To each phase correspond resources. Consequently, the interface synthesis during partitioning consists in determining the transfer unit resource (CPU / DMA), and scheduling the bus transfer resource ($Tbus$) and the hardware interface resource (blocking / FIFO). This generic model for software $\rightarrow$ hardware communications and three examples of resource implementation and bus scheduling are represented in figure 2. Hardware $\rightarrow$ software communications are similarly modeled.

The protocol and the cost of the communication are determined by the bus scheduling in the $[Ts, Tr]$ interval.
• $Tbus + d = Tr$: *blocking* protocol $\rightarrow$ control logic cost.
• $Ts = Tbus$: *non-blocking buffered* protocol $\rightarrow$ FIFO cost.

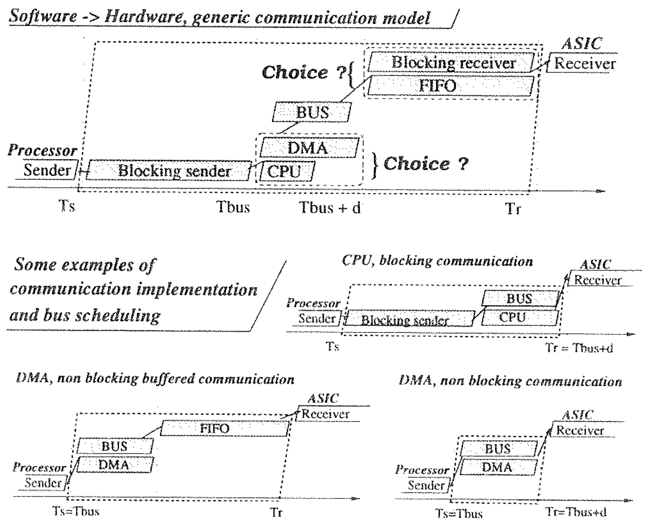

Figure 2. Resources of the communication models

• $Tbus + d = Tr, Tbus = Ts$: *non-blocking unbuffered* protocol $\rightarrow$ no cost.

Moreover, with this model, communication cost can be taken into account when calculating the sharing of communication resources. For example, if two bus resources have the same scheduling, the corresponding cost will be the cost of two buses. Alternatively, if bus accesses are sequenced, the cost will correspond to the cost of only one bus.

4. Partitioning Methodology

The methodology including communication is depicted in figure 3. This methodology is based on library elements characterized by a cost, an execution time and a set of resources. For instance, we only consider uniprocessor architectures.

Of course, when direct successors or direct predecessors of a communication node are not implemented, communication overhead is not taken into account. This is why we include communication nodes as "potential" nodes before partitioning.

We see three main advantages to treating communications in a homogeneous way, employing the same model for communications and operations:
• We do not need a specific algorithm to deal with communications. The HW/SW partitioning algorithm is well adapted to this.
• During partitioning, the codesign objectives (minimization of cost and execution time) are also applied to communications.
• The communication model is independent of the partitioning algorithm. This makes possible the use of multiple heuristics, each adapted to a different objective.

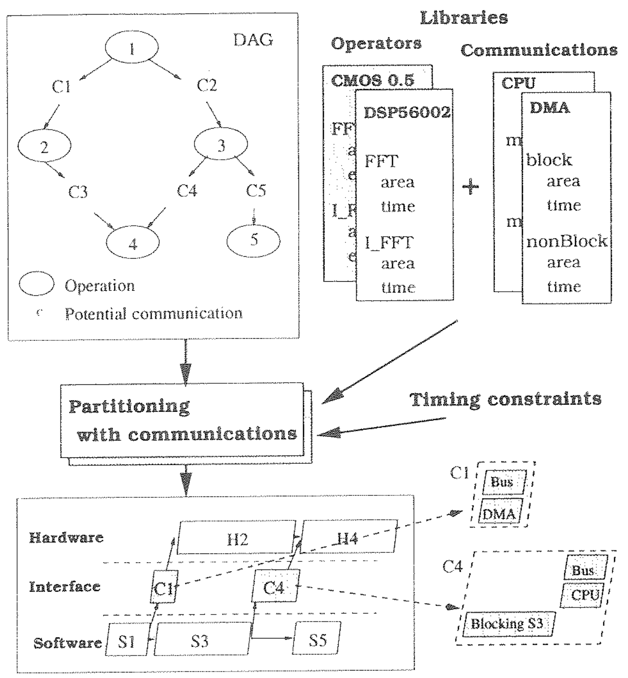

Figure 3. Methodology, including communications during partitioning

The only new restriction for the partitioning heuristics is that now a particular node ordering must be respected. The ordering captures the fact that the assignment and scheduling of a communication node cannot be done before the assignment and scheduling of its direct predecessor and direct successor nodes. Consequently the partitioning methodology has two stages within each iteration:

- First, choose assignment and scheduling for a node.
- Second, choose assignment and scheduling for the communication nodes whose successor and predecessor nodes have just been implemented during the first stage.

Partitioning iterations proceed until all nodes are allocated.

We apply this restriction to the heuristic we propose to solve the scheduling and assignment problems with resource optimization (refer to [11] for further details). Our heuristic is an extension of the Force-Directed scheduling algorithm, adapted to hardware-software partitioning.

For each node i and each possible time step j, a pair of forces, called *repel forces* ($F^j_{impl}(i)$), are calculated, one for the software implementation and one for the hardware implementation.

$$F^j_{impl}(i) = Self\_force^j_{impl}(i) + \sum_{k \neq i} Induced\_force(k)$$

The higher the repel force, the higher the cost. Each repel force is the sum of a $Self\_force$ and a total $Induced\_force$. The $Self\_force$ reflects the local cost of the implementation and the $Induced\_force$ represents the global cost. The total $Induced\_force$ for node i expresses the sum of the constraints induced by node i on all the other nodes. Once all the forces have been calculated, the heuristic chooses the implementation and schedule for the node that minimize the forces. The procedure continues until all nodes have been implemented. The heuristic in two stages is summarized below.

```
/* first stage */
while there are unimplemented nodes
  foreach implementation, scheduling for node $\neq$ coms
    $F_{impl}$ = Self_force() + $\sum$Induced_force()
  Choose Min ($F_{impl}$)
  Implement_schedule_node()
/* second stage */
  while there are new communication nodes
    foreach implementation, scheduling for node = coms
      $F_{impl}$ = Self_force() + $\sum$Induced_force()
    Choose Min ($F_{impl}$)
    Implement_schedule_communication_node()
```

5. Example: GMDFα

Our example is the implementation of the GMDFα (Generalized Multi-Delay Frequency Domain Filter) algorithm, used for acoustic echo cancellation to improve the quality of hand-free telephones. Its DAG is represented in figure 4 and a detailed description of its implementation is given in [1]. To partition this system, we use a DSP56002 for

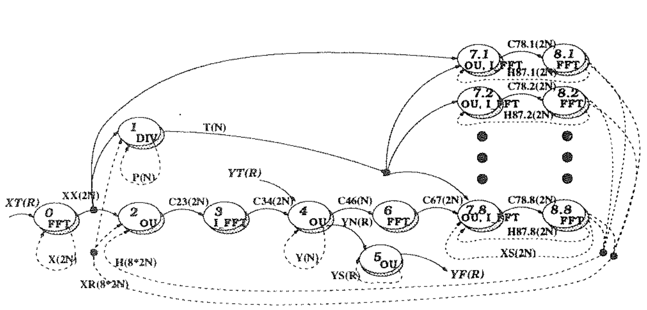

Figure 4. DAG of the GMDFα algorithm

the software part and an ASIC in 0.5μm CMOS technology for the hardware part. The maximum execution time for this system is $T_{max} = 6.25ms$, imposed by the sampling rate of the audio signal.

First, this system was partitioned with this T_{max} value and without any communication constraint. The execution time obtained for the system is $T_{syst} = 5.84ms$. Then, we synthesized communications with DMA/CPU optimization and we obtained $T_{syst} + T_{com} = 6.42ms$. However, this total execution time does not respect the timing constraint ($T_{max} = 6.25ms$). The results of this partitioning, DMA/CPU optimization (bold-face numbers), the protocol used for each transfer and the FIFO size for non-blocking

soft-hard	DMA	CPU	block	non-block	n-b. buff.
1←0	(80)	**50**		X	
4→6	(60)	**30**		X	
7.1←6	(50)	**50**		X	
7.i→8.i	**80*7**	(50*7)		X	
7.8→8.8	(50)	**50**		X	
8.i←2	(80*7)	**50*7**			14N
8.8←2	(80)	**50**		X	
no Optim.	940	930			14 N
optim.	**580** μs				**14 N**

Table 1. Communication synthesis after partitioning: $T_{syst} + T_{com}$ = 6.42ms, cost = 3.44mm^2

soft-hard	DMA	CPU	block	non-block	n-b. buff.
0→2	**80**	(50)		X	
6←4	**80**	(50)	**X**	⟸	(2N)
1→7.i	**50*8**	(30*8)			N
6→7.1	(80)	**50**		X	
6→7.i	**80**	(50)			2N
8.1←7.1	(80)	**50**		X	
8.i←7.i	**80*7**	(50*7)	**X**	⟸	(14N)
8.i←2	**80*7**	(50*7)			14N
8.8←2	**80**	(50)	**X**	⟸	(2N)
no Optim.	160	740			(35 N)
optim.	**100** μs				**17 N**

Table 2. Communication synthesis during partitioning: $T_{syst+com}$ = 5.15ms, cost = 4.17mm^2

protocol are shown in Table 1.

Second, the system was partitioned with our partitioning algorithm performing communication optimization. Results are shown in Table 2. Node assignment differs radically from the first partitioning and the total execution time becomes $T_{syst+com} = 5.15ms$. Thus, this execution time respects the timing constraint. With respect to node scheduling, non-blocking buffered transfers require FIFOs for 17N words (N=128 $\rightarrow$ 2176 words).

In order to quantify communication optimization, the system was partitioned again without any communication constraint but with a stronger timing constraint than the first partitioning: $T_{max} = 5.5ms$. We obtain the same node assignment than with the second partitioning but now node scheduling requires 35N words for non-blocking buffered transfers (data volumes are represented between parentheses in Table 2). Consequently our partitioning algorithm with communication optimization has replaced buffered communications by blocking communications thus removing the memorization cost (FIFOs) of 18N words.

Moreover, this partitioning shows us that the mix of CPU and DMA transfers divides the communication time respectively by 2 or 7 compared to the exclusive use of the DMA unit or the CPU unit.

6. Conclusion

We have shown the importance of hardware-software communications in telecommunication system codesign. More precisely, we have demonstrated the effect of communication on the partitioning task. We have described a model for communication which allows to partition systems by treating communications and operations in a homogeneous way. This results in an optimization taking into account communication cost, execution time, node assignment, node scheduling and resources. This methodology allows a single algorithm to carry out simultaneously the partitioning task and the interface definition. The main limitation of our methodology is that the communication nodes increase the size of the DAG to which partitioning is applied. Partitioning time could become prohibitive for large graphs. However, we follow an "open" methodology, allowing multiple partitioning algorithms and communication models. In particular, we plan to incorporate the shared-memory model in our communication library.

References

[1] M. Auguin, C. Belleudy, J. Bergé, L. Freund, G. Gogniat, M. Israel, and F. Rousseau. A codesign experiment in acoustic echo cancelation: Gmdfα. *Proc. of ISSS*, Nov. 1996.

[2] A. Bender. Design of an optimal loosely coupled heterogeneous multiprocessor system. *Proc of ED&TC96.*

[3] J.-M. Daveau, T. B. Ismail, and A. Jerraya. Synthesis of system-level communication by an allocation-based approach. *ISSS*, sept. 1995.

[4] D.Filo, D. Ku, C. Coelho, and G. D. Micheli. Interface optimization for concurrent systems under timing constraints. *IEEE Trans On VLSI systems*, sept. 1993.

[5] J. Gong, D. D. Gajski, and S. Bakshi. Model refinement for hardware-software codesign. *Proc. of ED&TC 96.*

[6] R. Gupta and G. D. Michelli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test*, sept. 1993.

[7] J.Henkel, R.Ernst, U. Holtman, and T. Benner. Adaptation of partitioning and high-level synthesis in hardware/software co-synthesis. *Proc. of ICCAD*, Nov. 1994.

[8] A. Kalavade. System-level codesign of mixed hardware-software systems. *Ph.D. Dissertation*, sept. 1995.

[9] S. Narayan and D. Gajski. Interfacing incompatible protocols using interface process generation. *Proc. of DAC*, 1995.

[10] P.Chou, R. Ortega, and G. Borriello. Interface co-synthesis techniques for embedded systems. *Proc of ICCAD 95.*

[11] F. Rousseau, J.-M. Bergé, and M. Israel. Hardware/software partitioning for telecommunications systems. *Proc. of COMPSAC'96*, Aug. 1996.

[12] D. Thomas, J. Adams, and H. Schmit. A model and methodology for hardware-software codesign. *IEEE Design & Test of Computers*, sept. 1993.

[13] T.-Y. Yen and W. Wolf. Communication synthesis for distributed embedded systems. *Proc. of ICCAD*, 1995.