HiPART: A New Hierarchical Semi-Interactive HW-/SW Partitioning Approach with Fast Debugging for Real-Time Embedded Systems

Thomas Hollstein, Jürgen Becker, Andreas Kirschbaum, Manfred Glesner
Darmstadt University of Technology
Institute of Microelectronic Systems
Karlstr 15, 64291 Darmstadt
{thomas|becker|glesner}@mes.tu-darmstadt.de

Abstract

In this contribution we present a new system-level hardware/software partitioning approach (HiPART) which is run in the frame of an integrated hardware software design methodology for embedded system design. The benefits of the approach result from an hierarchical partitioning algorithm, consisting of three phases of constructive and iterative methods. The main advantage of the system is a freely selectable degree of user interaction and manual partitioning. A permanent observation of timing constraint violations during partitioning guarantees the applicability for real-time systems.

1. Introduction

The scene of hardware/software codesign has introduced a number of hardware/software partitioning approaches to speed-up performance, to optimize hardware/software trade-offs, and to reduce total design time [6], [3], [11], [4], [12], [8], [7], [1] among others. The introduced approaches perform their techniques on different partitioning granularities, ranging from fine-grain [6], over medium-grain [3], [11], to coarse-grain [12], [7] granularities. These automated hardware/software partitioners use a variety of partitioning heuristics: extended greedy-algorithm [6], clustering [2],[9], simulated annealing [3], [7], dynamic programming [11] as well as a modified Kernighan/Lin heuristic in [12]. Input programs are partitioned among software and custom (reconfigurable) hardware parts (processors), as well as optimized embedded systems are designed.

As part of DICE (Darmstadt Interactive Codesign of Embedded Systems), the proposed HiPART hierarchical partitioning approach implements a set of communicating C and VHDL processes onto a heterogenous target hardware platform, consisting of different types of hardware modules (Microprocessors, DSPs, FPGAs, ASICs) being application-dependent connected in an optimized synthesized reconfigurable communication architecture [5], [10]. This library-based communication synthesis step improves overall system partitioning and design, since the corresponding informations are reflected in the cost function to be optimized during the partitioning and debugging process.

The introduced hierarchical partitioning method clusters closely related objects (instructions) in order to minimize communication and to handle increasing complexity. Such data-dependent clusters are analyzed due to their hardware and software performance in using pre-computed hardware and software performance (cost) values of all objects. The analyzed clusters are partitioned by a coarse-grained simulated annealing based heuristic. The obtained hardware/software partition is finally optimized in applying an extended Fiduccia/Mattheyses algorithm performing a "fine-tuning" on instruction-level granularity. Thus, the proposed hierarchical partitioning approach realizes a new hybrid fine-/coarse-grained partitioning strategy, also incorporating optimization with respect to subsequent synthesis of communication architectures.

Since the user has the option to move clusters from hardware to software and vice versa during the partitioning process, this approach offers the possibility to integrate system designers and/or application knowledge. Due to the incremental update technique of the cost function to be optimized, alternative partitions are generated rapidly. Thus, the introduced method (including graphical user interface) represents a step in direction of realizing fast debugging tools for hardware/software systems, comparable to software debugging. The development of such debugging tools seems to be necessary, because hardware/software codesign approaches should also include application information into the codesign process in order to obtain efficient results.

Initially this paper gives a brief description of the hardware/software partitioning problem. Section 3 presents the codesign system DICE briefly and its design flow. In section 4 the HiPART partitioning approach is described in detail. Section 5 discusses a computation-intensive application example followed by a summary in section 6.

2. Problem Definition

A promising design methodology for real-time embedded systems has to cope with several tasks in order to achieve an implementation which satisfies the system constraints. Mixed hardware/software systems are required if the targeted system performance can't be achieved by a pure software solution. The main design tasks in hardware/software codesign are hardware/software partitioning and the synthesis of communication structures. For validation of the functionality of the system at the actual design state, hardware/software cosimulation is applied at different levels of abstraction and rapid prototyping for the final implementation.

The system level functional partitioning problem can be defined as follows:

Definition 2.1 *partitioning problem*

Instance: *Given a partitioning graph $G = (V, E)$ consisting of a set $V = \{v_1, v_2, \ldots v_{n_V}\}$ of vertices and a set $E = \{e_1, e_2, \ldots e_{n_E}\}$, $E \subseteq V \times V$ of edges and an edge cost function $c : E \rightarrow \mathbb{R}^+$. Nodes v_i are functional objects (operations, processes or operation clusters) of a heterogeneous system specification. Edges e_i represent interrelations of nodes (control flow dependencies, data dependencies, common properties and relations of interest). Given a set $U = \{U_1, U_2, \ldots U_{n_P}\}$ of target units (U_1 is a software module, all other units are hardware modules), a set $AC = \{ac_2, \ldots, ac_{n_P}\}$ of unit hardware area constraints and a set $TC = \{tc_1, \ldots, tc_{n_{TC}}\}$ of timing constraints.*

Configurations: *All n_P-way partitionings $\mathcal{P} = \{P_1, P_2, \ldots P_{n_P}\}$ with $P_i \subseteq V$, and $\bigcup_{i=1}^{n_P} P_i = V$, and $P_i \cap P_j = \phi$ for $i, j \in \{1, \ldots, n_P\}, i \neq j$*

Solutions: *All partitionings $\mathcal{P} = \{P_0, P_1, \ldots P_{n_P}\}$ such that all $ac_i \in AC$ (with $i \in \{2, \ldots, n_P\}$) and all $tc_j \in TC$ (with $j \in \{1, \ldots, n_{TC}\}$) are satisfied*

Minimize: *$c(\mathcal{P}) = \sum_{i=1}^{n_E} c(e_i)$*

For performing partitioning constructive and iterative algorithms can be applied. In section 4 we introduce a combined partitioning method with close interactive designer involvement. The main challenge of hardware/software partitioning is to find a trade-off between algorithm runtime and appropriate cost functions. A new feature of the HiPART algorithm is the capability to guarantee to find solutions within a space bounded by predefined timing constraints.

3. DICE: HW/SW Codesign Environment

The design flow of the DICE system is shown in Fig. 1. A system specification, consisting of any number of concurrent VHDL and C processes is converted to a concurrent CDFG (CCDFG).

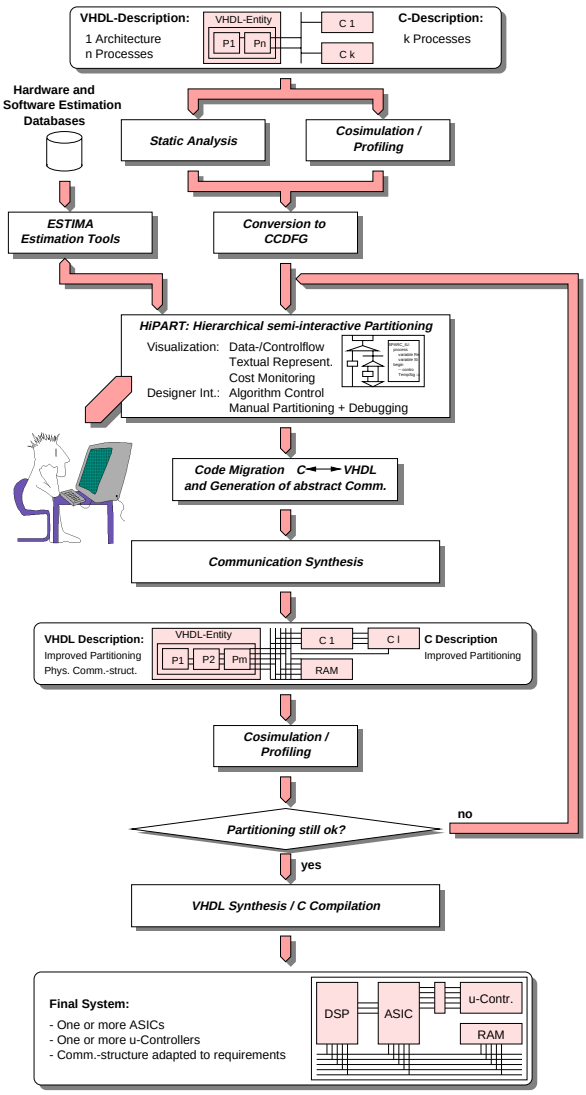

Figure 1. Design Flow in the DICE Codesign Environment

Communication in between of processes is done by abstract synchronous and asynchronous `send` and `receive` functions. During HW/SW cosimulation, profiling information is collected to be used for hardware and software performance estimations during partitioning. HW/SW partitioning is performed on the CCDFG based on a semi-interactive approach influenced by the designer using the HiPART graphical user interface. After HW/SW repartitioning code transformations C $\leftrightarrow$ VHDL are performed for code pieces with modified implementation attributes. During communication synthesis, abstract communication operators are replaced by connections using buses, buses with shared memory or FIFO buffers. In a subsequent cosimulation step a validation of the resulting functionality and the compliance with constraints is checked. If some con-

straints are violated, a HW/SW repartitioning with stronger constraints is necessary. If not, the code can be synthesized/compiled to target architectures as ASICs, FPGAs, μC or the DICE rapid prototyping platform REPLICA.

4. HW/SW Partitioning

HiPART is a new hierarchical hardware/software partitioning algorithm. An initial system specification, consisting of any number of concurrent C and VHDL processes, is converted into a CCDFG (concurrent control/dataflow graph). This graph is visualized by the HiPART graphical user interface (GUI) CCDFGView, which provides all functionality for interactive partitioning control. The structure of the HiPART partitioning environment is depicted in Fig. 2.

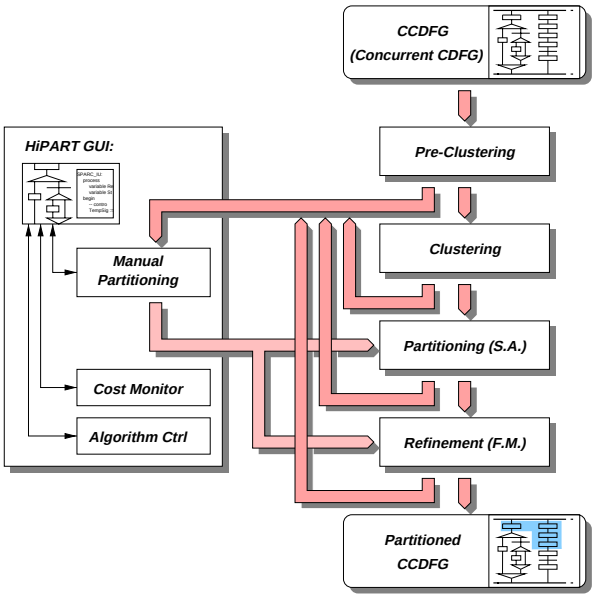

Figure 2. HiPART **Interactive Partitioning Methodology**

The HW/SW partitioning strategy is to keep as much as possible in software. Timing constraints may be attached by the designer using the GUI. The partitioning takes care of these constraints and hardware processes will be generated, if this is necessary to satisfy constraints. The basic partitioning granularity is fixed in a pre-clustering step. For CCDFGs with a large number of nodes, a reduction of complexity to a parametrizeable number of nodes n_{Cl} is achieved by clustering. The partitioning of resulting clusters (or CCDFG nodes) is performed by application of simulated annealing (s.a.). Since clustering reduces the optimization space, the s.a. may not find the absolute global cost minimum. Therefore additional optimization potential is explored by a refinement step, where clusters are iteratively resolved and post-partitioned using the Fiduccia-Mattheyses heuristic (still applying constraint observation). In between of each automatic partitioning step any manual partitioning may be applied via GUI. An absolute and differential cost monitoring of each step gives the designer the possibility to explore dependencies and to tune/debug the setting of algorithmic parameters efficiently.

4.1 Pre-Clustering

In the pre-clustering step the granularity of basic partitioning entities will be fixed. In classical partitioning approaches single operations or basic blocks are used as atomic elements for partitioning. Since operations may differ significantly with respect to hardware area and execution performance (e.g. for simple logic operators compared to full parallel multiplication) a general selection of basic block granularity will not lead to an optimal initial graph configuration. Furthermore designer knowledge cannot be included in systems with fixed granularity. Therefore we selected a *heterogeneous granularity approach*. First of all, an operation or basic block related automatic pre-clustering is executed. In an additional interactive step, the user may pre-cluster sets of nodes, which have to be mapped onto the same partition. Additionally operations, which may be realized in software only (e.g. pointer operations) will be pre-clustered automatically into one common partition.

4.2 Clustering

Clustering is applied in order to reduce the complexity of the subsequent simulated annealing partitioning step in terms of nodes. The maximum number of nodes is limited to a user-defined number n_{Cl} of cluster trees. By that the partitioning time is kept in reasonable limits. If the number of CDFG nodes is less than n_{Cl}, the clustering step has not to be performed.

The closeness function d between of two clusters is defined as follows:

$$\begin{aligned} d'(v_i, v_j) \quad = \quad & k_{comm} \cdot d_{comm}(v_i, v_j) \\ & + k_{share} \cdot d_{share}(v_i, v_j)(1 - d_{flex}(v_i, v_j)) \\ & + k_{flex} \cdot d_{flex}(v_i, v_j) \\ & + k_{initial} \cdot d_{initial}(v_i, v_j) \end{aligned}$$

k_{comm}, k_{share}, k_{flex} and $k_{initial}$ are user adjustable coefficients. d_{comm} is the communication closeness, containing all data dependencies in between of v_i and v_j multiplied by their profiling execution count and bitwidth. In order to enable hardware sharing, d_{share} is a measure for the similarity of operations contained in both clusters:

$$d_{share}(v_i, v_j) = \frac{\sum_{ot \in SharedOp} size(ot, \min\{w_{max}(ot, v_i), w_{max}(ot, v_j)\})}{\sum_{ot \in SharedOp} size(ot, w_{max}(ot, v_i \cup v_j))}$$

(with denominator zero check) where $w_{max}(ot, v)$ is:

$$w_{max}(ot, v) = max\{bitwidth(ot, n) | n \in v\}$$

d_{flex} is a user defined measure for the probability of later specification changes of the code (no changes expected: 0, probably changed: 1). Nodes with high flexibility should be partitioned into software. $d_{initial}$ is a measure for the degree of coincidence with the initial specification (HW or SW).

Algorithm 1 Clustering

initialize all nodes v_i of $G_{Cl}(V, E)$ by CDFG nodes
initialize edge heap: $H = \phi$
for all pairs of nodes $v_i, v_j \in V$ **do**
  $d(v_i, v_j)$ = ComputeCloseness(v_i, v_j)
  if $d(v_i, v_j) > 0$ **then**
    insert edge $\{v_i, v_j\}$ into heap H
  end if
end for
while $|V| > n_{Cl}$ **and** H not empty **do**
  select from H edge $\{v_i, v_j\}$ with maximum $d(v_i, v_j)$
  for all $v_k \in adj(v_i), k \neq i, k \in \{1, \ldots |adj(v_i)|\}$ **do**
    $H := H \setminus \{v_i, v_k\}$
  end for
  for all $v_k \in adj(v_j), k \neq j, k \in \{1, \ldots |adj(v_j)|\}$ **do**
    $H := H \setminus \{v_j, v_k\}$
  end for
  $v_{new} := v_i \cup v_j$
  properties(v_{new}) = MergeProperties(v_i,v_j)
  $V := (V \setminus \{v_i, v_j\}) \cup \{v_{new}\}$
  for all $v_k \in (adj(v_i) \cup adj(v_j)) \setminus \{v_i, v_j, v_{new}\}$ **do**
    $d(v_{new}, v_k)$ = ComputeClosess(v_{new}, v_k)
    if $d(v_{new}, v_k) > 0$ **then**
      insert edge $\{v_{new}, v_k\}$ into H
    end if
  end for
end while

The value of the closeness measure $d'(v_i, v_j)$ increases with the size of v_i and v_j. In order to be able to achieve an equalized cluster size distribution, the closeness is reduced as follows:

$$d(v_i, v_j) = \begin{cases} \frac{d'(v_i, v_j)}{\left(\frac{|v_i|+|v_j|}{N}\right)^{attenuation}} & |v_i| + |v_j| > N \\ d'(v_i, v_j) & else \end{cases}$$

with $N = threshold \cdot |V_{CDFG}|/n_{Cl}$. The values of $treshold$ and $attenuation$ can be parametrized.

The clustering algorithm itself is straightforward (Alg. 1) and follows the strategy to select clusters to be fused by closeness priority. In each clustering step a cost update to all clusters adjacent to the fused ones has to be performed.

4.3 Partitioning

The main partitioning step is done by application of simulated annealing (s.a.). Advantages of s.a. are the ability to to overcome local minima of the cost function and robustness with respect to modifications of the cost function. Given a partitioning graph $G_P(V, E)$ (resulting from clustering or manual prepartitioning), a set TC of timing constraints and a set AC of hardware area constraints, during partitioning a cost function $c(\mathcal{P}, G_P(V, E), TC, AC)$, which assigns a cost value to each possible partitioning, is minimized. In difference to definition 2.1 constraints are included in the cost function. Constraint violations will be punished by very high cost increase. Benefits of this concept are a unified cost function and the possibility of intermediate entrance into forbidden areas during optimization. The cost function is defined as follows:

$$\begin{aligned} c(\mathcal{P}, G_P(V, E), TC, AC) = \\ \sum_{p=P_2}^{P_{n_P}} (c_{area}(E, p) + k_{share} c_{share}(p)) \\ + \frac{1}{|V|} \sum_{v \in V \setminus P_1} k_{flex} c_{flex}(v) \\ + \frac{1}{|V|} \sum_{v \in V} k_{initial} c_{initial}(v) \\ + \sum_{e \in E} k_{comm} c_{comm}(e) \\ + \sum_{tc \in TC} c_{constr}(tc) \end{aligned}$$

k_{area}, k_{share}, k_{flex}, $k_{initial}$ and k_{comm} are coefficients which allow cost function adjustment by the designer. The area costs of *hardware* partitions are calculated as

$$c_{area}(E, p) = k_{area} area(E, p) + c_{area\_violation}(E, p)$$

with

$$area(E, p) = area\_op(p) + area\_wire(E, p)$$

$$c_{area\_violation}(E, p) = \begin{cases} 0 & area(E, p) < ac(p) \\ e^{2 \cdot (area(E,p) - ac(p))} - 1 & \text{else} \end{cases}$$

The area $area\_op(p)$ required for functional operators consists of separated contributions for shared operators (with rough multiplexer estimation) and non-shared operators. The wiring area is derived by a constant factor from the total hardware area. The uniformity of *shared* operations in *hardware partitions* is targeted by inclusion of the factor

$$c_{share}(p) = \frac{n_{types}(p)}{n_{types}}$$

where n_{types} is the number of shared operator types in the whole system specification. This contribution forces that operation types, which have to be shared due to designer selection are drawn into the same partition. The flexibility

measure generates costs for preliminary hardware clusters only and is computed based on the nodes n contained in a cluster v:

$$c_{flex}(v) = \frac{1}{|v|} \sum_{n \in v} flex(n)$$

Costs for deviation from initial specification type are included for clusters implemented in hardware or software:

$$c_{initial}(v) = \frac{1}{|v|} \sum_{n \in v} initial(n)$$

Communication costs are considered for inter-partition edges $e_{ext} = (v_i, v_j)$ by inclusion of:

$$c_{comm}(e) = \\ = \begin{cases} 0 & part(v_i) = part(v_j) \\ bitwidth(e) \cdot profiling(e) & \text{else} \end{cases}$$

Constraints are included with strong violation punishment by an exponential function (oc = over-constraining):

$$c_{constr}(tc) = \\ = \begin{cases} 0 & \text{if } oc(tc) \cdot d(path(tc)) \leq t_{max}(tc) \\ e^{k_{constr}(oc(tc) \cdot d(path(tc)) - t_{max}(tc))} - 1 & \text{else} \end{cases}$$

In the real implementation the exponential function is piecewise defined for a certain interval and then continued by a linear function with huge gradient (gradient is continued constantly from interval edge).

4.4 Refinement

During refinement initial clustering stages may be resolved iteratively combined with partitioning using a modified (complex cost function) Fiduccia-Mattheyses heuristic. Hereby a further local minimization of the cost function is achieved.

5. Partitioning Results

The HiPART algorithm has been applied to several applications: fuzzy controller, combustion engine control and a compress algorithm. The first two examples could be partitioned within seconds without clustering, due to a small number of nodes (<100). The partitioning has also been applied to C compress algorithm (868 nodes). Timing constraints have been set on the full execution time. The partitioning into a mixed HW/SW realization has been performed successfully within five minutes of CPU time (SPARC 20).

6. Conclusion

The paper presented a new hierarchical partitioning approach realizing a new hybrid fine-/coarse-grained partitioning strategy which allows any degree of user interaction. Due to fast incremental updates of the partitioning cost function values, the proposed method is a step in direction of debugging tools for hardware/software systems. According to the increasing complexity and heterogeneousness of applications, system designers and application information can be incorporated into the overall codesign process. In many cases this seems to be necessary to obtain efficient solutions.

DICE represents a hardware/software codesign system applicable to both: implementing complex applications onto a heterogeneous target hardware platform (incl. optimized synthesized communication architectures), and for rapid system prototyping of efficient implementation solutions for efficient systems-on-the-chip (SOCs).

References

[1] J. K. Adams and D. E. Thomas. The Design of Mixed Hardware/Software Systems. In *Proceedings of the Design Automation Conference*, pages 515–520, June 1996.

[2] E. Barros, W. Rosenstiel, and X. Xiong. A Method for Partitioning UNITY Language in Hardware and Software. In *Proceedings of the European Conference on Design Automation*, pages 220–225, Sept. 1994.

[3] R. Ernst, J. Henkel, and T. Benner. Hardware-Software Cosynthesis for Microcontrollers. In *IEEE Design & Test*, pages 64–75, Dec. 1993.

[4] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, 1994.

[5] M. Gasteier. *Cosimulation und Kommunikationssynthese im Entwurf gemischter Hardware/Software-Systeme*. PhD thesis, Darmstadt University of Technology.

[6] R. K. Gupta, C. N. Ceolho, and G. De Micheli. Program Implementation Schemes for Hardware-Software Systems. *IEEE Computer*, pages 48–55, Jan. 1994.

[7] R. W. Hartenstein and J. Becker. Two-Level Partitioning of Image Processing Algorithms for the Parallel Map-oriented Machine. In *Proceedings of the 4th Int. Workshop on Hardware/Software Co-Design CODES/CASHE '96*, Pittsburgh.

[8] J. Hou and W. Wolf. Process Partitioning for Distributed Embedded Systems. In *Proceedings of the Int. Workshop on HW/SW Codesign (CODES/CASHE))*, pages 70–76, Mar. 1996.

[9] T. B. Ismail, K. O'Brien, and A. Jerraya. Interactive System-level Partitioning with PARTIF. In *Proceedings of the European Design & Test Conference*, pages 464–468, Mar. 1994.

[10] A. Kirschbaum and M. Glesner. Rapid Prototyping of Communication Architectures. In *IEEE Workshop on Rapid System Prototyping*, pages 136–141, Chapel Hill, USA, June 1997.

[11] P. V. Knudsen and J. Madsen. A Dynamic Programming Algorithm for Hardware/Software Partitioning. In *Proceedings of the CODES/CASHE '96*, Pittsburgh, USA.

[12] F. Vahid. Modifying Min-Cut for Hardware and Software Functional Partitioning. In *Proceedings of the 5th Int. Workshop on Hardware/Software Co-Design CODES/CASHE '97*, Braunschweig, Germany.