Multi-way Clustering Techniques for System Level Partitioning

M. L. López Vallejo[†] and J. C. López López[‡]

† Dept. Ingeniera Electrónica
ETSI Telecomunicación, Univ. Politécnica Madrid
Ciudad Universitaria s/n, E–28040 Madrid, Spain
marisa@die.upm.es

‡ Dept. Informática
ESI, Univ. Castilla La Mancha
E–13071 Ciudad Real, Spain
lopez@uclm.es

Abstract— **This paper presents a new approach to system partitioning based on clustering techniques. The adaptation of the heuristic is based on a robust system model. Major modifications include the formulation of a general and effective closeness function and the use of a new control scheme. The final implementation has been widely checked, obtaining promising conclusions.**

I. Introduction

System partitioning deals with the assignment of parts of a system description to heterogeneous implementation units: ASICs (hardware), standard or embedded micro-processors (software), memories, etc. This is a key task in system level design, because the decisions that are made at this time directly impact on the performance and cost of the final implementation. During this stage, many different aspects must be considered to tackle the heterogenity of the different processing units. Consequently, the automation of the partitioning phase is excepcionally complicated.

To automate system level partitioning many algorithms and techniques have been developed in different co-design environments [4], [3], [5]. For instance, previous work has been done in the adaptation of classic circuit partitioning algorithms (clustering [1], mincut [9]), the use of general optimization methods (simulated annealing [3], tabu search [2], linear programming [7], etc.) or genetic algorithms [8].

In this paper we propose the introduction of system level issues within hierarchical clustering algorithms to partition complex systems into hardware and software components. The application of these techniques at system level requires significant modifications of the classical algorithm, because the resulting partitions have different nature. Hierarchical clustering has been previously used for Hw-Sw partitioning. In [10] different and interesting closeness metrics are defined, but their application is not straight-forward and their use is not clearly described. We believe that the closeness metrics must be different for the hardware and the software partitions. Thus, the clustering procedure needs to be modified and specific metrics for the different implementation units must be defined. Additionally, we have defined a robust problem model which helped us introducing important modifications to the original algorithm. In [1], other closeness metrics are introduced, being mainly based on the specification language UNITY. Nevertheless, cost issues are not considered, as we propose in this work.

This work is funded by CICYT project TIC2000-0583-C02

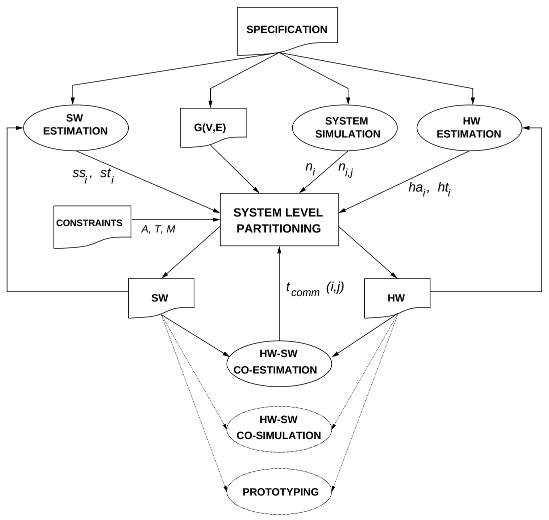

Fig. 1. Flow of Information in the Partitioning Model.

The structure of this paper is as follows. First, the particular system partitioning problem we solve is presented. Next section describes the proposed clustering scheme in depth. Finally, experimental results will be presented and some conclusions will be drawn.

II. System Partitioning

The resolution of a problem requires the definition of a model representing all the important issues related to the specific problem. We describe in this section the system partitioning problem we deal with.

A. Partitioning Model

The flow of information within the partitioning process presented here is depicted in Figure 1. The input is an execution flow graph which comes from the initial system specification. In this graph (directed and acyclic), nodes stand for basic computation units and edges represent data and control dependencies. Nodes can be big pieces of information (tasks, processes, etc.) or small ones (instructions, operations), following respectively a coarse or fine granularity approach.

Every graph vertex, v_i, is labeled with several attributes: Hw area (ha_i), Hw execution time (ht_i), Sw memory size (ss_i), Sw execution time (st_i) and the average number of times the task is executed (n_i). Edges have also associated a weigth value that represents the associated communication delay, $t_{com}(i,j)$.

As it is well known, system partitioning is clearly influenced by the target architecture where the Hw and the Sw will be mapped. The target architecture we consider consists of one standar processor, several hardware co-processors and a shared memory accessed through a common bus. Interface modules are used to connect the processor and the co-processors to the bus.

The output of the partitioning tool is not only an assignment of blocks to Hw or Sw implementations, but also their scheduling and the communication cost produced in the Hw-Sw interface. Scheduling is performed by means of a list-based scheduling algorithm. The scheduler takes into account the timing estimates of every vertex in the graph and the dependencies among them. As output, it gives the design latency, T_p, and the communication overhead.

The validity of the proposed solution is measured by means of significant design quality attributes, like the Hw area, A_p, the design latency, T_p and the required memory space, M_p. Every attribute is associated to a design constraint, being these constraints the maximum allowed area, A, the muximum execution latency, T, and the size of the available memory, M.

Here we must introduce other important parameters, the *extreme values*. These parameters are obtained through the extreme implementations, the *all-hardware* and the *all-software* solutions. These bounds provide an idea of the difficulty of finding a solution with a given set of constraints, ensuring that we do not look for an impossible solution. From the *all-hardware* solution we obtain, $MinT$, the minimum design latency and $MaxA$, the maximum Hw area. From the *all-software* solution we obtain two more parameters: $MaxT$, the maximum design latency and $MaxM$, the maximum memory space. To ensure that we do not look for an impossible solution, the system constraints must always verify: $0 \leq \sum A \leq MaxA$, $0 \leq M \leq MaxM$ and $MinT \leq T \leq MaxT$.

B. Problem Formulation

More formally, the Hw-Sw partitioning problem can be formulated as follows. Given a system description in the form of a task graph, directed and acyclic, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges, with a timing goal (for instance a latency T), a target architecture $\mathcal{D} = (\Pi, \Gamma_1, \Gamma_2, \dots, \Upsilon)$ (where Π stands for the standard processor, Γ_i represents the i-th hardware co-processor and Υ is the interface model) accompanied by a set of architectural constraints (A, maximum Hw area, M, maximum memory size, B, bus transfer rate and W, bus width), and a cost function that evaluates the quality of a given solution $\mathcal{F} : \mathcal{G} \to [0, \infty)$; the Hw-Sw partition, $\mathcal{P}$, is a function that assigns every vertex of $\mathcal{G}$ to a processing unit of $\mathcal{D}$ with a starting time $t \in [0, \infty)$ while minimizing the cost function $\mathcal{F}$. Formally:

$$\mathcal{P} : \mathcal{V} \to \{\Pi, \Gamma_1, \Gamma_2, \dots \Gamma_n\} \times [0, \infty) \;\; /$$
$$\begin{cases} \forall v \in \mathcal{V} \quad \mathcal{P}(v) = (i,t) \text{ with } \begin{cases} i \in \{\Pi, \Gamma_1, \dots \Gamma_n\} \\ t \in [0, \infty) \end{cases} \\ \mathcal{F} : G \to [0, \infty) \quad \text{is minimized} \end{cases}$$

It must also be verified that:

1. $\forall v_1, v_2 \in \mathcal{V} \quad \mathcal{P}(v_1) \neq \mathcal{P}(v_2)$: *condition of time-space exclusion.*

2. $\forall v_1, v_2 \in \mathcal{V} / v_1 < v_2$, ($<$ expresses an order relation) *if* $\mathcal{P}(v_1) = (i, t_1)$ and $\mathcal{P}(v_2) = (j, t_2) \Rightarrow t_1 + t(i, v_1) < t_2$, *condition of data dependencies among vertices*, where $t : \mathcal{D} \times \mathcal{V} \to [0, \infty)$ is a function that provides for every vertex $v \in \mathcal{V}$, its execution time in the processing unit $i \in \{\Pi, \Gamma_1, \dots \Gamma_n\}$.

3. If $v_{end} \in \mathcal{V} \; / \; \forall v_j \in \mathcal{V}, \; v_j \neq v_{end}, \; v_j < v_{end}, \; \mathcal{P}(v_{end}) = (k, t_{fin}) \quad \Rightarrow \; t_{fin} + t(k, v_{end}) \leq T$, *condition of adjustment of time goal.*

4. Let Φ_{hw} be $\Phi_{hw} = \{v_j \in \mathcal{V}, \; \mathcal{P}(v_j) = (k, t_j) \; / \; k \in \{\Gamma_1, \dots \Gamma_n\} \; \}$; it must be verified that $\alpha(\Phi_{hw_i}) = A_{p_i} \leq A_i, \; 1 \leq i \leq n$, *condition of adjustment of the area constraint*, with α being the function that estimates the co-processor Hw area with relation to Φ_{hw} in a given point of the design space.

5. Let Φ_{sw} be $\Phi_{sw} = \{v_j \in \mathcal{V}, \; \mathcal{P}(v_j) = (k, t_j) \; / \; k = \Pi \; \}$; it must be verified that $\mu(\Phi_{sw}) = M_p \leq M$, *condition of adjustment of the memory size*, with μ being the function that evaluates the memory space needed to run a software code with relation to Φ_{sw}.

C. Hierarchical Clustering

Hierarchical clustering is a constructive method which groups pairs of partitioning objects based on a proximity value between the objects. For every algorithm iteration all distances among partitioning objects must be evaluated, and the pair of objects with the maximum proximity value is grouped to form a cluster. As clustering proceeds, new clusters are created by grouping two individual objects or by grouping an object or cluster with another cluster. The process is iterated until a single cluster is produced and a hierarchical cluster has been formed.

The algorithm is fully characterized by:

- The closeness function that computes the proximity values.
- The cut level in the cluster tree that provides the required number of partitions (clusters).

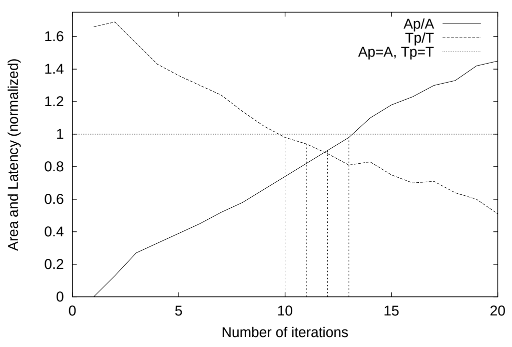

Fig. 2. Evolution of the area and latency attributes.

Both issues must be strongly modified to perform system partitioning, because of the different nature of the target processing units. It is obvious that a closeness metric used to group objects that will run in a standard processor cannot be suitable for those objects implemented with dedicated hardware. The proposed closeness function takes into account the system model information grouping those partitioning objects with a clear tendency to be implemented as hardware. The process will be driven by the system constraints.

C.1 Control Scheme

Every iteration the algorithm selects the two objects with the best time improvement when implemented as hardware (the highest latency decrease). Objects not grouped along the process will be assigned to the software processing unit. The cut level is dynamically computed whenever a cluster is grown taking into account the system constraints. It can be said that this is a *"software-oriented"* approach, since all objects are supposed to reside originally in software and during the process they are extracted to the hardware processing unit. With this procedure, the whole cluster tree does not need to be built.

When a cluster is grown, the area and latency constraints (A_i, T) are checked. A_i is the current *active Hw area constraint*. All Hw area constraitns are ranked form the lowest to the highest. Initially the lowest constraint is selected as active. If one of the clusters grows over the active area constraint, that cluster is marked as completed (it cannot grow any more) and the next constraint in the rank is marked as active. The tree construction stops whenever:

1. The time constraint is met ($T_p < T$) and the Hw area of the biggest cluster is bellow the active costraint ($A_{p_i} < A_i$).
2. The whole Hw area constraint is exceeded ($\sum A_{p_i} > \sum A_i$) while the latency constraint is not yet met ($T_p > T$).

In the first case, area and latency constraints are met and the algorithm is almost finished (only the memory constraint must be checked). In the second case, the area constraint is exceeded without fulfilling the timing objectives, and therefore, a refinement phase is necessary. This can be accomplished by a local search procedure working with objects with smaller grain (thus, the last clusters must be broken up).

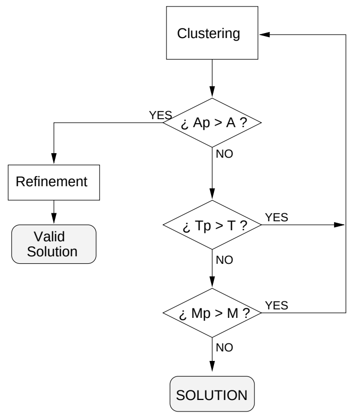

Fig. 3. Control scheme of the clustering process

This approach is based on the idea that to find a solution there must be several points in the design space satisfying their respective constraints, as can be seen in the figure 2. This plot shows the evolution of the area and latency quality attributes normalized with respect to the design constraints while building the whole cluster tree in a clustering process. Axis X shows the number of iterations in the clustering process (number of clusters grown, $|\mathcal{V}+1|$). As the number of clusters added to the tree increases, the Hw area becomes larger and the design latency should decrease. When time and area constraints are satisfied, the memory constraint should be checked, playing a secondary role, though. If memory constraint is met, the algorithm finishes. Otherwise the cluster growth continues while other (primary) constraints are still under their limits. Figure 3 illustrates the algorithm control flow described above.

C.2 Closeness Function

The proposed closeness function takes into account the system model information, using the estimates associated with the vertices and edges of the system graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. As it is well known, the exchange of information among the different processing units of the architecture penalizes the design latency. Closeness metrics should intend to reduce this problem. In addition, since the algorithm follows a *"hardware extraction"* approach the time improvement obtained when extracting an object to hardware should also be considered. The expression we have formulated as a closeness function that takes into account both effects is the following:

$$C_{i,j} = q_T(n_i\frac{\Delta t_i}{st_i} + n_j\frac{\Delta t_j}{st_j}) + q_C\frac{t_{com}(v_i,v_j)}{t_{r\_com}(v_i) + t_{r\_com}(v_j)} \quad (1)$$

where $\Delta t_i = st_i - ht_i$ represents the time improvement obtained when the object i is moved from software to hardware. The function $t_{com}(v_i, v_j)$ computes the communication between nodes v_i and v_j in the interface following a general model of the architecture. The weight factors q_T and q_C help the designer emphasize which factor he wants to optimize.

As can be seen, every function term has been normalized with its Sw time, in such a way that the resulting closeness value is greater for those objects with bigger difference between hardware and Sw execution times. The communication term has been normalized with respect to the addition of communication values of the cluster object with other objects of the system graph, being its expression the following:

$$t_{r\_com}(v_i) = \sum_{v_k \in \mathcal{V}} t_{com}(v_i, v_k) \ / \ \exists\, (v_i, v_k) \in \mathcal{E} \quad (2)$$

The use of this kind of normalization highlights the fact that the communication between two design modules must be taken into account only when there are many transfers just between these modules. This term is then less important when communication with the rest of the design modules is considerable.

When checking the plausibility of this closeness function several deficiencies in the behavior of the algorithm were observed. In some cases the algorithm grouped objects with very high communication values and a good time improvement, but with a considerable size. This halted the algorithm early because of the Hw area constraint. For this reason the closeness function was modified to cluster objects that had the previous time and communication considerations but which required little Hw area. The resulting function expression is:

$$C_{i,j} = q_T(n_i\frac{\Delta t_i}{st_i} + n_j\frac{\Delta t_j}{st_j}) + q_C\frac{t_{com}(v_i,v_j)}{t_{r\_com}(v_i) + t_{r\_com}(v_j)} + q_A\frac{n_o\frac{MaxA}{|\mathcal{V}|}}{ha_i + ha_j} \quad (3)$$

where a Hw area term has been introduced, controlled by its corresponding weight factor q_A. This term has a clear meaning: its value is greater when the area of the resulting cluster is smaller than the average system area. This average area is computed by dividing the area of the *all-hardware* solution, $MaxA$, by the total number of vertices of the system graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ given by the cardinal of $\mathcal{V}$, $|\mathcal{V}|$. The parameter n_o is the number of nodes integrating the cluster, $n_o = |i| + |j|$. As will be described, each clustering object (single or composed) will be characterized with the same attributes as the nodes of the system model, ha_i, ht_i, etc. This formulation allows us to handle objects with different size or various components in a uniform way.

In the same way we can introduce a term for considering the memory space. In this case, since we are extracting modules to hardware, the memory term must try to group objects with large-sized memory. This value is also computed using the parameter $MaxM$, memory of the *all-software* solution. The final expression for the closeness function for two objects, i, j, is:

$$C_{i,j} = q_T(n_i\frac{\Delta t_i}{st_i} + n_j\frac{\Delta t_j}{st_j}) + q_C\frac{t_{com}(v_i,v_j)}{t_{r\_com}(v_i) + t_{r\_com}(v_j)} + q_A\frac{n_o\frac{MaxA}{|\mathcal{V}|}}{ha_i + ha_j} + q_M\frac{ss_i + ss_j}{n_o\frac{MaxM}{|\mathcal{V}|}} \quad (4)$$

C.3 Cluster Characterization

It is important to remark that the resulting clusters must exhibit the same characteristics as the basic objects, in such a way that no accumulative error is introduced. This characterization allows us to use the same closeness function throughout the algorithm's execution. Consequently, we define the following approximation to characterize a cluster $k = i \cup j$:

• The resulting Hw area is computed by adding up of the Hw area of the nodes that integrate the cluster.

$$ha_k = \sum_{v_n \in p_i} ha_n + \sum_{v_m \in p_j} ha_m \quad (5)$$

This estimation is quite rough, because resource sharing is not considered. Nevertheless the approach is completely valid, and we leave for future study the improvement of the area estimation procedures.

• The cluster memory space is computed as the sum of the memory size of its composing vertices.

$$ss_k = \sum_{v_n \in p_i} ss_n + \sum_{v_m \in p_j} ss_m \quad (6)$$

• The Sw execution time of the cluster is the sum of the Sw execution time of its composing nodes, due to code serialization.

$$st_k = \sum_{v_n \in p_i} st_n + \sum_{v_m \in p_j} st_m \quad (7)$$

• Hw execution time can be computed in two ways:

1. If there are no data dependencies among the cluster vertices, concurrency is possible, and the Hw execution time is given by:

$$ht_k = max\{ht_n \text{ with } v_n \in p_i \cup p_j\} \quad (8)$$

TABLE I
RESULTS OBTAINED AFTER APPLYING THE PROPOSED CLUSTERING SCHEME TO THE EXAMPLES.

	Characterization					Constraints			Results			
Graph	nodes	MaxA	MinT	MaxT	MaxM	A	T	M	A_p	T_p	M_p	Cost
LU	9	69315	74	454	19595	15000	400	17000	9619	356	16463	0.556
						30000	200	16000	25604	193	9757	0.606
						50000	150	12000	34714	143	5018	0.536
FFT	15	106355	145	842	33619	*30000*	*400*	*25000*	*28764*	*557*	*18633*	*23.89*
						60000	250	25000	59054	265	7880	1.573
						80000	200	15000	78192	172	3526	0.575
DCT	9	21952	2231	7312	329692	16000	4000	100000	10976	3675	75013	0.556
						13000	5500	200000	5488	5372	164125	0.502
						20000	3000	120000	13720	2825	48779	0.529
DCT16	36	34944	6712	20680	1143982	10000	20000	800000	4368	16240	776552	0.472
						20000	15000	800000	7644	14500	628607	0.483
						30000	16000	750000	5460	15660	721591	0.444
Laplace	9	73009	79	386	17811	*20000*	*250*	*16000*	*17132*	*274*	*9013*	*2.024*
						30000	200	15000	23921	213	7631	1.243
						50000	200	12000	37263	141	3679	0.536
Mean	9	132626	99	607	27244	*60000*	*300*	*20000*	*43865*	*386*	*15438*	*13.01*
						80000	500	10000	76137	260	9698	0.538
						120000	200	10000	103603	176	4676	0.570

2. If there are data dependencies the cluster vertices must be scheduled. Since at every algorithm iteration a system scheduling is performed, we know the starting and finishing times of all the vertices, and therefore, the cluster execution time is:

$$ht_k = max\{t_{end}(v_n)\} - min\{t_{end}(v_n)\} - t_{idle}, \quad (9)$$

where $v_n \in p_i \cup p_j$ and t_{idle} is the time the Hw co-processor is idle between $min\{t_{end}(v_n)\}$ and $max\{t_{end}(v_n)\}$. This kind of timing evaluation has two clear advantages. First, there is no accumulative error, because ht_k is recalculated at every step of the algorithm. Second, the computational complexity is not greater, since we take advantage of the scheduling performed to evaluate design constraints.

Regarding the cluster edges, we will only consider the edges coming in and out of the cluster, canceling the edges within the cluster vertices. The external edges keep their original attributes, because the cluster will be considered exactly as a new system vertex.

III. EXPERIMENTAL RESULTS

The algorithm has been applied to a set of real examples, enumerated in table I, and to a set of generated examples with bigger size. Here we will focus on results for a simple architecture (a single hardware co-processor) to compare the performance of the clustering algorithm to other partitioning procedures.

Table I shows first the characterization of the examples: the size of the system graph and the values of the *extreme implementations*. Area is given in arbitrary units, time in ns and memory in bytes. Hard constraints are printed with italic face. To measure every solution quality (column *Cost*) we have used the following cost function [6]:

$$\begin{aligned}\mathcal{F}(\mathcal{P}) = k_A\frac{A_p}{A} + k_T\frac{T_p}{T} + k_M\frac{M_p}{M} + k_a r(A, A_p) \\ + k_t r(T, T_P) + k_m r(M, M_P)\end{aligned} \quad (10)$$

where the weight factors are $k_a = 0.3$, $k_t = 0.4$, $k_m = 0.3$ and $k_{c_i} = 150$ and the function r is a penalty function that corresponds to (for the area constraint):

$$r(A, A_p) = \max\ \{0, \frac{[A_p - A]}{A}\} \quad (11)$$

This function helps us interpreting the results. Since $\sum_i k_i = 1$ then $\mathcal{F}(\mathcal{P}) = 1$ is a figure of merit because (1) constraint overheads will produce cost values much greater than 1, (2) attribute values tuned to the constraints will produce costs close to the unity, and, (3) in the case that the solution could be optimized the cost value will be lower than 1.

A first analysis of the results shows that for hard constraints the algorithm cannot find a valid solution. Even worse, this algorithm provides solutions quite far from the valid region. In these cases the area constraint (a low value compared with $MaxA$) halts the process when latency constraint is not met. If the constraints are not too hard, the method can provide a workable solution. The main advantage of the approach is that it is very fast. We have compared with other partitioning techniques: simulated annealing, min-cut and an expert system. Figure 4 shows the cost obtained when running all these partitioning procedures with the examples of table I. We have represented the cost of the solutions obtained for each example which has been checked with four sets of constraints represented in the figure as four points in the axis X. These four set of constraints have been ordered from harder to softer constraints.

TABLE II

Sobel example: results for different system constraints in both implementations.

A	T	M	A_p	T_p	M_p	CPU(s)	Cost	A_p	T_p	M_p	CPU(s)	Cost
30000	500	45000	25835	837	31829	0.890	68.9727	35773	601	36753	31.71	12.7475
30000	600	45000	25835	837	31829	0.889	24.1513	31830	667	32213	30.07	3.35964
30000	700	45000	25835	837	31829	0.883	6.43341	29705	707	36042	24.428	0.93675
40000	500	40000	35245	777	27226	1.043	46.836	40237	546	27538	29.832	2.21998
50000	500	30000	46097	639	22012	1.391	12.326	49683	483	22582	23.629	0.91031

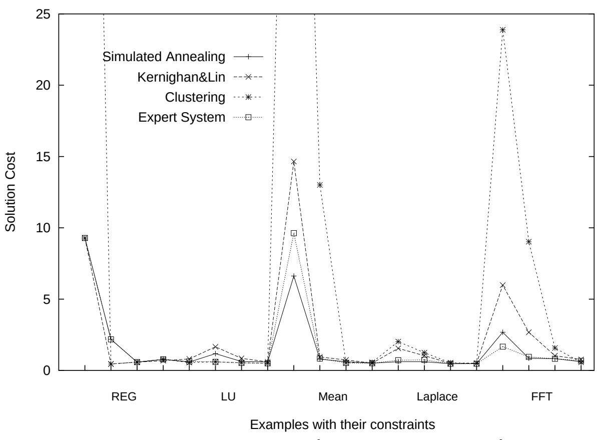

Fig. 4. Cost values obtained for the examples of table I.

To solve the problem of the bad behavior of the algorithm for hard constraints we have used the group migration algorithm to refine the clustering results. Table II shows the results obtained after running this new version with a 20 vertices example and different hard constraints. For this example the extreme values are $MaxA = 132020$, $MinT = 219$, $MaxT = 1134$, $MaxM = 53386$ ($q_A{=}q_T{=}q_M{=}q_C{=}1$). It can be clearly seen that the clustering algorithm is blocked in the same solution because of the low Hw area constraint. This problem is solved by the refinement stage, what results in an increase of the partitioning execution time. However, the computation time of this mixed procedure is smaller than the time required by the simulated annealing implementation. In most cases the combination of the two algorithms resulted in a very good solution: similar to one obtained with simulated annealing in shorter time.

Concerning the algorithm execution time, for all the tests we performed the fastest technique was always the clustering algorithm. This is due to the fact that clustering only performs system scheduling when a cluster is grown. The other techniques must schedule the design at each trial.

As conclusion, hierarchical clustering can be used to perform a fast design-space exploration, specially if a refinement stage is performed later. This method can work with very large system graphs due to its short computation time. Consequently, we recommend its use for fine grain descriptions as a pre-partitioning stage. After this step the design space is reduced and other techniques can work with objects of different granularity. We have obtained excellent results with the clustering technique followed by Kernighan&Lin.

IV. Conclusions

We have shown that classical clustering techniques provide attractive alternatives to implement system partitioning. The heuristic adaptation has been based on a robust system model. Major modifications include the formulation of a general and effective closeness function and the use of a new control scheme.

The final implementation has been widely checked. The algorithm provides good results with very short execution times. However, the hardness of the constraints biases the quality of results. Solutions for this problem have been proposed.

Future work covers the implementation of multi-stage clustering after decomposing the proposed closeness function and the application of the proposed technique to more complex architectures.

References

[1] E. Barros, W. Rosenstiel, and X. Xiong. HW/SW Partitioning with UNITY. In *Handouts of 2nd International Workshop on HW-SW Codesign*, Oct 1993.

[2] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System Level Hardware/Software Partitioning based on Simulated Annealing and Tabu Search. *Design Automation for Embedded Systems*, 2(1):5–32, January 1997.

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

[4] R. K. Gupta and G. De Micheli. HW-SW Cosynthesis for Digital Systems. *IEEE Design & Test of Computers*, pages 29–41, 1993.

[5] A. Kalavade and E. A. Lee. The Extended Partitioning Problem: Hw/Sw Mapping, Scheduling and Implementation-bin Selection. *Journal of Design Automation of Embedded Systems*, 2(2):125–164, March 1997.

[6] M.L. López Vallejo, J. Grajal, and J.C. López. Constraint-driven System Partitioning. In *Proc. DATE'00*, pages 411–416, March 2000.

[7] U. N. Shenoy, P. Banerjee, and a. Choudhary. A system-level synthesis algorithm with guaranteed solution quality. In *Proc. DATE'00*, pages 417–424, March 2000.

[8] V. Srinivasan, S. Radhakrishnan, and R. Vemuri. Hardware Software Partitioning with Integrated Hardware Design Space Exploration. In *Proc. DATE'98*, pages 28–35, Paris, France, 1998.

[9] F. Vahid. Modifying Min-Cat for Hardware and Software Functional Partitioning. In *Proc. Workshop on HW/SW Co-Design CODES/CASHE'97*, Mar 1997. Braunschweig, Germany.

[10] F. Vahid and D. D. Gajski. Clustering for Improved System-level Functional Partitioning. In *Proc. ISSS'95*, pages 28–33, 1995.