Restricted Range Exhaustive Search: A New Heuristic for HW/SW Partitioning of Task Graphs

Bastian Knerr, Martin Holzer, and Markus Rupp
Institute of Communications and RF Engineering
Vienna University of Technology, Austria
Email: {bknerr,mholzer,mrupp}@nt.tuwien.ac.at

Abstract— **This paper presents a new deterministic algorithm for the classical HW/SW partitioning problem, targeting a common architecture model consisting of a microprocessor and a set of ASIC blocks. The system to be partitioned is represented as an acyclic task graph, for which a set of constraints on makespan, area, and code size exist. The restricted range exhaustive search algorithm is introduced that is well suited for the subset of human made task graphs, since it exploits their typical characteristics such as locality, sparsity, and their degree of parallelism. The performance of the new algorithm is demonstrated in comparison to a genetic algorithm, a tabu search, and the global criticality/local phase algorithm.**

I. Introduction

Hardware/software (HW/SW) co-design is a design paradigm for the joint specification, design and synthesis of mixed HW/SW systems. The interest in automatic co-design techniques is driven by the increasing diversity and complexity of applications employing embedded systems, the need to curb the rising costs of design, verification, and test for such systems, as well as reducing time-to-market. At numerous times during the design process crucial decisions have to be made that dramatically influence the quality and the cost of the final solution. Some design decisions might have an impact of even 90% of the overall cost; among these HW/SW partitioning is of prominent relevance [1], [2].

HW/SW partitioning is generally defined as the mapping of functional parts of the system description to architectural components of the platform, while satisfying a set of constraints like time, area, power, throughput, delay, etc. HW then usually addresses the implementation of a functional part, e.g. performing a finite impulse response filter or Walsh-Hadamard transform, as a dedicated HW unit features a high throughput and can be very power efficient. On the other hand, such a custom data path is much more expensive to design and inflexible when it comes to future modifications. Contrarily, SW addresses the implementation of the functionality as code to be compiled for a general-purpose processor or digital signal processor (DSP). It generally provides flexibility and is cheaper to maintain, whereas the required processors are more power consuming and offer less performance in speed. The optimal trade-off between cost, power, performance, and chip area over the complete system has to be identified. For a sound introduction into system partitioning in electronic system design please refer to the literature [1], [2]. The task of partitioning has been thoroughly researched and enhanced over the last 15 years and produced a number of feasible solutions, which depend heavily on their prerequisites: the architecture model, the communication model, the granularity of the functional objects, etc. A short overview of the most relevant work in this field is given in Section II.

In this work a new deterministic algorithm is introduced that addresses the classical binary HW/SW partitioning problem. The chosen scenario follows the work of other well-known research groups in the field, like the GCLP algorithm by Kalavade and Lee [3], Wiangtong et al. [4], and Chatha and Vemuri [5]. The fundamental idea behind the presented strategy is the exploitation of distinct graph properties like locality and sparsity, which are very typical for human-made designs. Generally speaking, the algorithm *locally* performs an exhaustive search of a restricted size while incrementally stepping through the graph structure. The algorithm shows strong performance compared to implementations of the genetic algorithm as used by Mei [6], the *penalty reward* tabu search proposed by Wiangtong [4], and the GCLP algorithm of Kalavade [3].

The rest of the paper is organised as follows. Section II lists the most reputed work in the field of partitioning/scheduling techniques. Section III illustrates the basic principles of system partitioning, gives an overview of typical graph representations, and introduces the common platform abstraction. It is followed by a detailed description of the proposed algorithm and an identification of the essential graph properties in Section IV. In Section V the sets of test graphs are introduced and the results for all algorithms are discussed. The work is concluded and perspectives to future work are given in Section VI.

II. Related Work

Heuristic approaches dominate the field of partitioning algorithms, since partitioning is known to be an intractable problem in most formulations [7]. Genetic algorithms have been extensively used [6], [8], as well as simulated annealing [9], [10]. To a smaller degree tabu search [11] and greedy algorithms [12] have also been applied. Other research groups developed custom heuristics such as the early work in [13] or the GCLP algorithm, which features a very low algorithmic

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

complexity [3].
With respect to combined partitioning and scheduling approaches, the work in [10] has to be mentioned. The approaches in [6], [14] also add communication events to links between HW units and SW functions. The architecture model varies from having a single SW and a single HW unit [9], [12], which might be reconfigurable [5], to a limited set of several concurrently running HW units combined with a general-purpose processor [4].

III. System Partitioning

This section covers the fundamentals of system partitioning. Due to limited space only a general discussion of the basic terms is given in order to ensure a sufficient understanding of our contribution. For a detailed introduction to partitioning, please refer to the literature [1], [2].
In embedded system design the term *partitioning* combines two tasks: *allocation*, i.e. the selection of architectural components, and *mapping*, i.e. the binding of system functions to these components. Usually a number of requirements, or *constraints*, are to be met in the final solution, for instance execution time, area, throughput, power consumption, etc. This task is known to be an intractable optimisation problem [15], in some formulations NP-hardness has been proven [3], [7]. The system functionality is typically abstracted into a graph $G = (\mathcal{V}, \mathcal{E})$ representation. In Figure 1a, six vertices $\mathcal{V} = \{a, .., f\}$ are depicted which are connected by six edges $\mathcal{E} = \{e_1, .., e_6\}$. The vertices cover the functional objects of the system, or *processes*, whereas the edges mirror data transfers between different processes. Depending on the granularity of the graph representation, the vertices may stand for a single operational unit (MAC, Add, Shift) or have the rich complexity of an MPEG decoder. The majority of the partitioning approaches [3]–[5], [14] decide for medium sized vertices that cover the functionality of FIRs, IDCTs, a quicksort, or similar procedures. Every vertex has

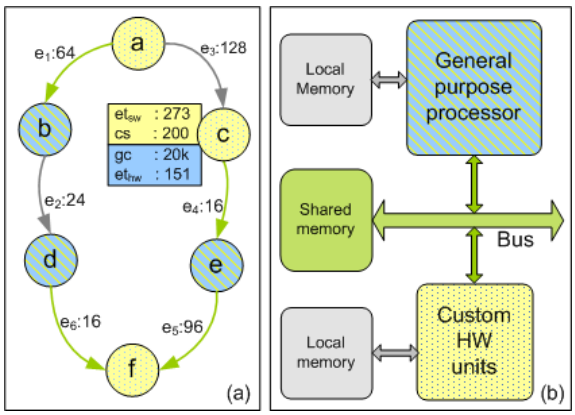

Fig. 1. *(a) Process graph, annotated with characteristic values. (b) Typical platform model.*

been annotated with characteristic values that, e.g. in the case of the GCLP algorithm for the binary (SW, HW) partitioning problem, build a quadruple: (*execution time* (et_{sw}) *and code size* (cs) *for SW, execution time* (et_{hw}) *and area in gates* (gc) *for HW*). This scenario has been chosen to be the base of the comparison to the other partitioning techniques. The edges are annotated with the number of data samples (bits) transmitted per invocation of one process. The mapping of the task graph to the given architecture in Figure 1b is performed with the objective to meet constraints for the makespan, area, and code size. The platform model features a general purpose processor, which allows for sequential execution of the assigned processes, and an FPGA or a set of ASICs for a custom data path, which allows for concurrent execution of the assigned processes. Since in this work the achievable system time is considered as one of the key system traits, for which constraints exist, a reliable feedback on the makespan of a distinct partitioning solution is obligatory. Therefore, we adhere to a detailed communication model, that provides collision-free schedules for all partitioning solutions seen. Table I provides the example access times for reading and writing bits via the different resources of the platform in Figure 1b. Communication of processes on the same resource uses preferably the local memory, unless its capacity is exceeded. Processes on different resources use the system bus to the shared memory. This detailed communication model is

Communication	read (bits/μs)	write (bits/μs)
Local sw memory	512	1024
Local hw memory	2048	4096
Shared system bus	1024	2048
Direct I/O	4096	4096

TABLE I

Maximum throughput for read/write accesses to the communication/memory resources.

similar to Wiangtong's work [4]. The underlying scheduling technique can be switched between static and dynamic versions of Hu's priority level schedule [16], Hwang's earliest task first schedule [17], and Knerr's LEP schedule [18].

A. Cost and Constraints

As can be seen in Equation 1, the cost function is a weighted linear combination of the characteristic values for makespan, area, and code size, due to its simple and easily extensible structure. The quality of the obtained solution, the cost value Ω_P for the solution P, is then:

$$\Omega_P = p_T(T_P)\ \alpha\ \frac{T_P - T_{min}}{T_{limit} - T_{min}} + p_A(A_P)\ \beta\ \frac{A_P}{A_{limit}} + p_S(S_P)\ \xi\ \frac{S_P}{S_{limit}}. \quad (1)$$

Here, $T_P \leq T_{limit}$ is the makespan of the graph for partitioning P; $A_P \leq A_{limit}$ is the sum of the area of all processes mapped to hardware; $S_P \leq S_{limit}$ the sum of the code sizes of all processes mapped to software. With the weight factors α, β, and ξ the designer can set individual priorities. If not stated otherwise, these factors are set to 1. If

$T_P > T_{limit} \vee A_P > A_{limit} \vee S_P > S_{limit}$ is true, a penalty function is applied to enforce solutions within the limits:

$$p_A(A_P) = \begin{cases} 1.0 & , A_P \leq A_{limit} \\ \left(\frac{A_P}{A_{limit}}\right)^{\eta} & , A_P > A_{limit} \end{cases} \quad (2)$$

p_T and p_S are defined analogously. If not stated otherwise, η is set to 4.

The validity percentage $\Upsilon = N_{valid}/N$ is the quotient of the number of valid solutions N_{valid} divided by the number of all solutions N, for a graph set containing N different graphs.

The constraints are specified by three ratios R_T, R_A, R_S to give a better understanding of their strictness. The ratios are obtained by the following equations:

$$R_T = \frac{T_{limit} - T_{min}}{T_{max} - T_{min}}, R_A = \frac{A_{limit}}{A_{max}}, R_S = \frac{S_{limit}}{S_{max}}. \quad (3)$$

The values A_{max}, S_{max}, and T_{max} are simply built by the sum of the gate counts gc, code sizes cs, and maximum execution time et_{max} of every process (plus the maximum transfer time tt_{max} of every edge), respectively. The computation of T_{min} is obtained by scheduling the graph under the assumption of unlimited FPGA/ASIC resources and no conflicts on any sequential device. Thus, T_{min} and T_{max} are respectively the lower and upper bounds on the makespan.

The computational runtime Θ has been measured in clock cycles by the high resolution timer *QueryPerformanceCounter* of the MS Windows API on a PC (AMD Athlon 64 3000+, 1.8GHz Processor).

IV. The RRES Algorithm

A thorough review of typical system graph structures has been performed to isolate a relevant subset of task graphs. Concretely, the complete industry design of a UMTS baseband receiver chip [19] has been analysed. Many contributions incorporate one or two example designs taken from development worksuites they lean towards [3], [9], while others introduce a fixed set of typical and very regular graph types [4]. Nearly all of the mentioned approaches generate additional sets of random graphs to obtain a reliable fundament for test runs of their algorithms. However, random graphs, if not further specified, can differ dramatically from human made graphs. Graphs in electronic system design, in which programmers capture their understanding of the functionality and the data flow, can be identified by a set of specific properties, of which three are of importance for this new algorithm.

Human made system graphs with a granularity of functions like FIR, FFTs, etc. have a strong affinity to rather short edges, i.e. vertices pass data and parameters predominantly to vertices with a similar rank level. A descriptive metric, the rank-locality r_{loc}, is introduced:

$$r_{loc} = \frac{1}{|\mathcal{E}|} \sum_{e_i \in \mathcal{E}} \text{rank}(v_{sink}(e_i)) - \text{rank}(v_{source}(e_i)). \quad (4)$$

For any edge the rank difference of its vertices is calculated, summed and averaged. Unlike the classical k-locality property, the *rank*-locality is efficiently computable beforehand to characterise the system graph under consideration. Typical values are $r_{loc} < 3$ and very often even $r_{loc} \approx 1$.

The second property of interest is the sparsity, $\rho = \frac{|\mathcal{E}|}{|\mathcal{V}|}$ of a graph. If the number of edges scales with the number of vertices, $|\mathcal{E}| \sim |\mathcal{V}|$, a graph is considered sparse, and if $|\mathcal{E}| \sim |\mathcal{V}|^2$, it is considered dense. Typical values are $\rho < 5$.

The degree of parallelism $\gamma = \frac{|\mathcal{V}|}{|\mathcal{V}_{CP}|}$, where $|\mathcal{V}_{CP}|$ is the number of vertices on the critical path, has typically low to medium values as well, $\gamma \in [2, \sqrt{|\mathcal{V}|}]$, for human made structures. Note, that in graphs with execution times for vertices or transfer times for edges, these times are usually taken as weights per vertex and per edge for the computation of γ.

Definitions for rank, γ, ρ, and k-locality can be found in the literature [20].

In Figure 2, a typical graph with low values for ρ and r_{loc} and is depicted. The rank levels are annotated at the bottom of the graphic. The following paragraphs describe the new

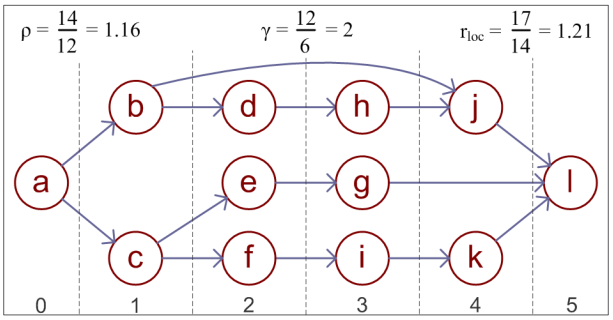

Fig. 2. Task graph with characteristic values for ρ, r_{loc}, and γ.

strategy to exploit such a graph structure. The fundamental idea is that in general a local optimal solution, here covering the rank levels $0-1$ does probably not interfere with an optimal solution for the rank levels $4-5$. Instead of finding proper cuts in the graph to ensure such a non-interference, which is very rarely possible, we consider a moving window (i.e. a contiguous subset of vertices) over the topologically sorted graph, and apply exhaustive searches on these subsets. The window is moved incrementally along the graph structure from the start vertices to the exit vertices. The precise algorithmic steps are described in the rest of the section.

The preparation phase of the algorithm comprises several necessary steps to boost the performance of the proposed strategy. A crucial part is certainly the identification of the order, in which the vertices are *visited* by the moving window and the construction of the initial solution. The latter is not in the focus of this work and is thus omitted in the interest of brevity. Constructional heuristics for the initial solutions in this setting can be found in the literature [21]. For the vertex order a vector is instantiated holding the vertices' indices. The main requirement for the ordering is that adjacent elements in the vector mirror the vicinity of readily mapped processes in the schedule. Different schemes to order the vertices have

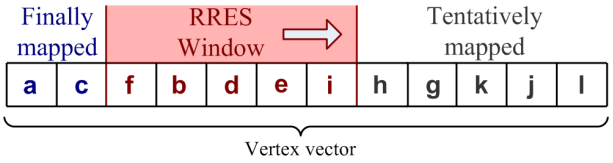

Fig. 3. Moving window for the restricted range exhaustive search on an ordered vertex vector.

been tested: a simple rank ordering completely neglects the annotated execution and transfer times; an ordering according to ascending Hu priority levels incorporates the critical path of every vertex; a more elaborate approach is the generation of two schedules, *as soon as possible* and *as late as possible*. For some vertices $v \in \mathcal{V}$ we obtain the very same start times $st(v) = st_{asap}(v) = st_{alap}(v)$, for both schedules, for others these are different, then $st(v) = \frac{1}{2}(st_{asap}(v) + st_{alap}(v))$. An alignment according to ascending values of $st(v)$ yielded the best results among these three schemes, since the dynamic range of possible schedule positions is hence incorporated. It has to be stated, that in the case of the binary partitioning problem exactly two different execution times for any vertex exist, and three different transfer times for the edges (hw-sw, hw-hw, sw-sw). In order to achieve a single value for execution and transfer times, again different schemes are possible: utilising the values from the initial solution, simply calculating their average, or utilising a weighted average, which incorporates the constraints. The last technique yielded the best results on the applied graph sets. The exact weighting is given in the following equation:

$$\begin{aligned} et = \frac{1}{3}(& R_S\ et_{sw} + (1 - R_S)et_{hw} + \\ & R_A\ et_{hw} + (1 - R_A)et_{sw} + \\ & \hat{R}_T\ et_{sw} + (1 - \hat{R}_T)et_{hw}\ ), \end{aligned} \tag{5}$$

where $\hat{R}_T = R_T$, if $et_{sw} \geq et_{hw}$, and $\hat{R}_T = 1 - R_T$ otherwise.

Once the vertex vector has been generated, the main algorithm starts. In Listing 1 pseudocode is given for the basic steps of the proposed algorithm. Lines $1-2$ cover the steps already explained in the previous paragraphs. The loop in lines $4-6$ is the windowing across the vertex vector with window length $|W|$.

Listing 1. Pseudocode for the RRES scheduling algorithm

```
0  RRES() {
1    createInitialSolution();
2    createOrderedVector();
3
4    for(i=1; i <= |V|-|W|; i++) {
5      exhaustiveSearch(i, i+|W|);
6    }
7  }
8
9  exhaustiveSearch(int v_i, int v_j) {
10   swapVertex(v_j);
11   for (int i=0; i < 2^(|W|-1); i++) {
12     createMapping(v_i, v_j, i);
13     if (constraints fulfilled)
14     { valid = true; }
15
16     if (cost < bestCost)
17     { storeSolution(); }
18     if (cost < bestValCost && valid)
19     { storeValidSolution(); }
20   }
21   mapVertex(v_i, bestSolution);
22 }
```

From within the loop, the exhaustive search in line 9 is called with parameters for the window $v_i - v_j$. The swapping of most recently added vertex v_j in line 10 is necessary to save half of the run time, since all solutions for the previous mapping of v_j have already been calculated in the iteration before. This is related to the break condition of the loop in following the line 11. Although the current window length is $|W|$, only $2^{|W|-1}$ have to be calculated anew in every iteration. In line 12, the current mapping according to the binary representation of loop index i is generated. In lines $13-19$, the checks for the best and the best valid solution are performed. The actual final mapping of the *oldest* vertex in the window v_i takes place in line 21. Here, the mapping of v_i is chosen, which is part of the best solution seen so far. When the window reaches the end of the vector, the algorithm terminates.

V. Results

Naturally, the crucial parameter of RRES is the window length $|W|$, which has strong effects on both the run time and the quality of the obtained solutions. In Figure 4, the first result is given for the graph set with the least number of vertices $|\mathcal{V}| = 20$, since a complete exhaustive search (ES) over all 2^{20} solutions is still feasible. The graph set contains 180 different graphs, with $r_{loc} = [1 \dots 7]$, $\rho = [1 \dots 6]$, and $gamma = [2 \dots 8]$. The constraints are rather strict to produce a portion of invalid results, $(R_T, R_A, R_S) = (0.4, 0.4, 0.5)$. The vertical axes show the range of the validity percentage Υ and the best obtained cost values Ω averaged over the 180 graphs. Over the possible window lengths $|W|$, shown on the

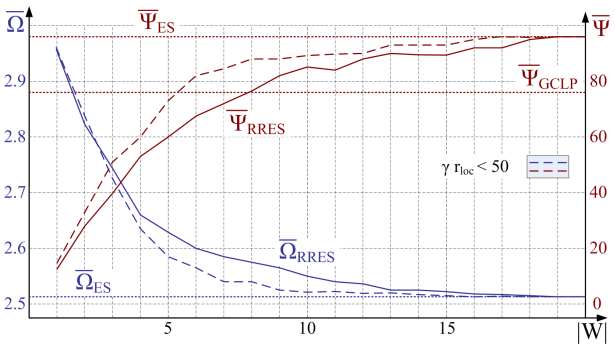

Fig. 4. Validity Υ, and cost Ω for RRES, GCLP and ES plotted over the window length $|W|$.

x-axis, the performance of the RRES algorithm is plotted. The dotted lines show the ES and GCLP performance. The

cost $\bar{\Omega}_{GCLP} = 3.22$ is not depicted, since it is outside the graph range. For a window length of 20, the obtained values for RRES and ES naturally coincide. The algorithm's performance is scalable with the window length parameter $|W|$. The trade-off between solution quality and run time can hence directly be adjusted by the number of calculated solutions $S = (|\mathcal{V}| - |W|)2^{|W-1|}$. The dashed curves are the cost and validity values over the graph subset, for which the product of rank locality and parallelism is $\gamma\, r_{loc} < 30$. Obviously, there is a strong dependency between the proposed RRES algorithm and this product. In the last part of this section, this relation is brought into sharper focus. The relatively low performance of the GCLP algorithm is understandable, since it is basically a greedy search and designed to have a very low algorithmic complexity $\mathcal{O}(|\mathcal{V}|^2)$ for sparse graphs. Due to its internal structure, the GCLP run time is comparable to RRES with $|W| = 3 \ldots 4$. However, with respect to the obtained quality of solutions, it cannot compete with more time consuming strategies like Genetic Algorithm (GA), Tabu Search (TS) or RRES.

For the following considerations, the parameter set of the GA implementation is briefly outlined. For a detailed description of the GA terms, please refer to the literature [22]. The chromosome coding utilises as fundament the same ordered vertex vector as depicted in Figure 3. Every element of the chromosome, a gene, corresponds to a single vertex. Possible gene values, or alleles, are 1 for HW and 0 for SW. Two selection schemes are provided, binary fighting and roulette wheel selection, of which the first showed better convergence. Mating is performed via two-point crossover recombination. Mutation is implemented as an allele flip with a probability 0.03 per gene. The population size is set to $2|\mathcal{V}|$, and the termination criterion is fulfilled after $2|\mathcal{V}|$ generations without improvement. These GA mechanisms have been selected according to GA implementations in literature [6], [8].

The third algorithm to benchmark against is the *penalty reward* TS from Wiangtong et al. [4]. It combines a short term memory, i.e. the tabu list of recently visited regions of the search space, with a long term memory, such that frequently visited regions of the search space are penalised (*diversification*), and regions that frequently return high-quality solution are rewarded (*intensification*). An aspiration criterion is provided, which ensures that globally best solutions are accepted, even when they are flagged tabu. According to their work, we chose an identical parameter set: neighbourhood size is $S_N = \lfloor\sqrt{|\mathcal{V}|/2}\rfloor$, number of tabu degrees $N_{td} = \lfloor\sqrt{|\mathcal{V}|/2}\rfloor$, so that the obtained tabu list length is $L_T = S_N N_{td} = |\mathcal{V}|/2$. The long term memory covers a range of 10 vertices, corresponding to sufficient memory for 2^{10} different regions. The termination criterion is fulfilled after $4|\mathcal{V}|$ iterations without improvement.

Table II contains selected information about the performance of the four algorithms on all graph sets. Note, that GA and TS contain randomised mechanisms. As a consequence their output for different runs on the same graph varies.

$\lvert\mathcal{V}\rvert$	GA			GCLP		
20	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$
strict	**2.52**	0.17	**92.2%**	3.22	0.26	68.6%
loose	2.07	0.11	100%	2.78	0.17	88.1%
	TS			RRES ($\lvert W\rvert = 10$)		
	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$
strict	2.56	0.19	83.4%	2.56	0.18	85.5%
loose	2.09	0.11	100%	**2.06**	0.12	100%
$\lvert\mathcal{V}\rvert$	GA			GCLP		
50	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$
strict	2.76	0.19	81.6%	3.17	0.11	72.2%
loose	2.21	0.12	100%	2.72	0.10	97.5%
	TS			RRES ($\lvert W\rvert = 10$)		
	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$
strict	2.77	0.19	77.5%	**2.70**	0.18	**93.3%**
loose	2.23	0.12	100%	**2.17**	0.11	100%
$\lvert\mathcal{V}\rvert$	GA			GCLP		
100	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$
strict	2.88	0.19	56.4%	3.89	0.38	21.8%
loose	2.28	0.12	100%	2.83	0.17	92.6%
	TS			RRES ($\lvert W\rvert = 10$)		
	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$	$\bar{\Omega}$	$\bar{\sigma}$	$\bar{\Upsilon}$
strict	2.81	0.20	62.4%	**2.70**	0.17	**99.4%**
loose	2.22	0.12	100%	**2.16**	0.12	100%

TABLE II

RESULTS OBTAINED FOR THE FOUR ALGORITHMS ON 180 DIFFERENT GRAPHS.

Hence, 30 runs per graph have been performed with GA and TS each. Their mean and standard deviation incorporate 180×30 values, as opposed to 180 for the GCLP and RRES algorithms. Table II shows the averaged results for all graphs with the sizes $|\mathcal{V}| = 50, 100$. The termination criteria of GA and TS, and the window length of RRES had been adjusted, so that their run times did not differ by more than 25%. The best values are shown in bold. On first inspection, the results expose advantages for RRES both in cost and validity for these graph structures. The GCLP algorithm trails by more than $25\% - 50\%$ in Ω and Υ, but is 50 to 100 times faster. Consequently, this algorithm is a reasonable candidate for very large graphs $|\mathcal{V}| > 200$, because only then its very low run time proves truly advantageous.

A more sophisticated picture of the algorithms' performance can be obtained if we plot the averaged cost values of GA, TS, and RRES of all graphs and all sizes over their rank-locality metric r_{loc} and the parallelism metric γ, respectively. Recall, that we identified typical system graphs in the field to have rather low values for r_{loc} and low to medium value for γ, while having low values for ρ. The product $\gamma\, r_{loc}$ has been calculated for all the sample graphs, and the performance of GA, TS, and RRES have been plotted against this characteristic value, as shown in Figure 5. The $\overline{\Omega}$ values are normalised to the minimum cost value $\bar{\Omega}_{min} = \min(\bar{\Omega}_{GA}, \bar{\Omega}_{TS}, \bar{\Omega}_{RRES})$, so that the performance differences are given in percentages above 1.00. For low values of $\gamma\ r_{loc}$, the RRES algorithm yields significantly better results, up to 7%. Its performance degrades slowly until it drops back behind GA for values larger than 50 and

behind TS for values larger than 80. Interestingly, GA loses performance as well, for values larger than 130. For clarity it has to be mentioned, that the majority of the graphs lies in the range $\gamma\ r_{loc} = [30, 80]$. This behaviour becomes clear,

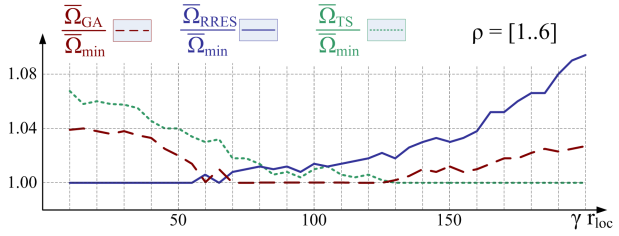

Fig. 5. Dependency between the product $\gamma\ r_{loc}$ and the obtained averaged cost values for GA, TS and RRES.

when we recall the intricacies of its chromosome coding. Due to the fundamental schema theorem [22] of genetic algorithms, adjacent genes in the chromosome coding have to reflect a corresponding locality in the system graph. With growing values of $\gamma\ r_{loc}$, the mapping of the graph onto the two-dimensional chromosome vector decreasingly mirrors the vicinity of the vertices within the graph. Hence, GA proves very sensitive to badly fitting chromosome codings, whereas TS remains insensitive to high values of the product $\gamma\ r_{loc}$. Consequently, it can be stated that a thorough a-priori analysis of the problem graph returning r_{loc}, ρ, and γ, all of which are derivable with linear algorithmic complexity for directed acyclic graphs, is an obligatory step before the designer can decide for an appropriate partitioning technique.

VI. Conclusions

In this work a new heuristic for the binary HW/SW partitioning problem has been introduced. A thorough analysis of its behaviour related to graph properties revealed a strong performance for a distinct subset of system graphs typical in the field of electronic system design. For this subset, the proposed RRES algorithm outperforms three other popular partitioning techniques based on the concept of genetic algorithms, Wiangtong's *penalty reward* tabu search, and the well reputed GCLP algorithm of Kalavade and Lee.
Future work will scrutinise the runtime of the RRES algorithm by revising the incremental movement of the RRES window. It may be possible to identify situations, in which more than one vertex could be mapped per movement of the window. Another interesting idea is the implementation of a short term memory for the moving window, in which the implementation type of vertices is fixed precociously, due to their contribution to the recently found global best solutions. This research is a mandatory step to address the extended partitioning problem, when there are more than two possible implementation alternatives per vertex.

References

[1] G. de Micheli, R. Ernst, and W. Wolf, *Readings in HW/SW Co-Design*. San Francisco, CA, USA: Morgan Kaufman Publishers, Academic Press, 2002.

[2] P. Marwedel, *Embedded System Design*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2003.

[3] A. Kalavade and E. Lee, "The Extended Partitioning Problem: HW/SW Mapping, Scheduling, and Implementation-Bin Selection," *Readings in HW/SW Co-Design*, pp. 293–312, 2002.

[4] T. Wiangtong, P. Cheung, and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in HW/SW Codesign," *Design Automation for Embedded Systems*, vol. 6, pp. 425–449, 2002.

[5] K. Chatha and R. Vemuri, "MAGELLAN: Multiway HW/SW Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs," in *Proc. of the 9th Int. Symposium on HW/SW Codesign (CODES)*. New York, NY, USA: ACM Press, 2001, pp. 42–47.

[6] B. Mei, P. Schaumont, and S. Vernalde, "A HW/SW Partitioning and Scheduling Algorithm For Dynamically Reconfigurable Embedded Systems," in *Proc. of ProRISC*, 2000.

[7] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic Aspects of HW/SW Partitioning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 136–156, 2005.

[8] V. Srinivasan, S. Radhakrishnan, and R. Vemuri, "HW/SW Partitioning with Integrated Hardware Design Space Exploration," in *Proc. of the Conf. on Design, Automation and Test in Europe (DATE)*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 28–35.

[9] J. Henkel and R. Ernst, "An Approach to Automated HW/SW Partitioning Using a Flexible Granularity Driven by High-Level Estimation Techniques," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 2, pp. 273–290, 2001.

[10] M. Lopez-Vallejo and J. Lopez, "On the HW/SW Partitioning Problem: System Modeling and Partitioning Techniques," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 3, pp. 269–297, 2003.

[11] P. Eles and Z. Peng and K. Kuchcinski and A. Doboli, "System Level HW/SW Partitioning Based on Simulated Annealing and Tabu Search," *Design Automation for Embedded Systems*, vol. 2, pp. 5–32, 1997.

[12] J. Grode, P. V. Knudsen, and J. Madsen, "Hardware Resource Allocation for HW/SW Partitioning in the LYCOS System," in *Proc. of the Conf. on Design, Automation & Test in Europe (DATE)*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 22–27.

[13] R. Gupta and G. De Micheli, "HW/SW Cosynthesis for Digital Systems," *Readings in HW/SW Co-Design*, pp. 5–17, 2002.

[14] R. Dick and N. Jha, "MOGAC: A Multiobjective Genetic Algorithm for the Co-synthesis of HW/SW Embedded Systems," in *Proc. of the IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 522–529.

[15] J. Hromkovič, *Algorithmics for Hard Problems*, 2nd ed. New York, NY, USA: Springer-Verlag, Inc., 2004.

[16] T. C. Hu, "Parallel Sequencing and Assembly Line Problems," Operations Research, Tech. Rep. 6, 1961.

[17] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM J. Comput.*, vol. 18, no. 2, pp. 244–257, 1989.

[18] B. Knerr, M. Holzer, and M. Rupp, "A Fast Rescheduling Heuristic of SDF Graphs for HW/SW Partitioning Algorithms," in *Proc. of IEEE Conf. on Communication System, Software and Middleware*, New Delhi, India, January 2006.

[19] W. Haas, M. Hofstaetter, T. Herndl, and A. Martin, "UMTS baseband chip design," in *Informationstagung Mikroelektronik*, Vienna, October 2003, pp. 261–266.

[20] R. Sedgewick, *Algorithms in C++, Part 5: Graph Algorithms*, 3rd ed. Addison-Wesley, January 2002.

[21] B. Knerr, M. Holzer, and M. Rupp, "Improvements of the GCLP Algorithm for HW/SW Partitioning of Task Graphs," in *Proc. of the 4th IASTED Int. Conf. on Circuits, Signals, and Systems (CSS)*, November 2006.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.