EXTENDING THE GCLP ALGORITHM FOR HW/SW PARTITIONING: A DETAILED PLATFORM MODEL AND PERFORMANCE IMPROVEMENTS

B. Knerr, M. Holzer, and M. Rupp

Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389,
1040 Wien, Austria

ABSTRACT
HW/SW partitioning of modern heterogeneous systems, which combine signal processing as well as multimedia applications, is usually performed on a task or process graph representation. As this optimisation problem is known to be NP-hard, existing partitioning techniques rely on heuristic methods to traverse the vast search space. The Global Criticality/Local Phase (GCLP) algorithm, initially introduced by Kalavade and Lee as an integral part of the Ptolemy work suite, has been frequently referred to as fast and powerful technique to generate high quality solutions for a combined partitioning/scheduling problem. Although having a good reputation, GCLP neglects essential information with respect to the underlying communication model. A detailed communication model for a typical System-On-Chip (SoC) architecture is introduced that considers different read and write times for all memory and bus resources. The internal mechanisms of the GCLP algorithm have been thoroughly analysed and several modifications are proposed that lead either to a significant increase of the quality of the obtained solutions without affecting the computation time of the algorithm or to a substantially lower computation time while increasing the output of valid partitioning solutions.

KEY WORDS
HW/SW Partitioning, Multi-Resource Scheduling, Design Automation, Optimization

1. INTRODUCTION

Modern system design, especially in the wireless domain, has to face hard challenges with respect to chip area, power consumption, and execution time while time-to-market is critical. The diversity of the requirements has led to extremely heterogeneous system architectures, whereas the short design cycles boosted the demand for early design decisions, such as architecture selection and HW/SW partitioning on the highest abstraction level, i.e. the algorithmic description of the system. HW/SW partitioning can in general be described as the mapping of the interconnected functional objects that constitute the behavioural model of the system onto a chosen architecture model. The task of partitioning has been thoroughly researched and enhanced during 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 Sec. 2.

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

One of the leading research groups to address the difficulties in modern system design established the Ptolemy Project (1991 - now) at the University of California, Berkeley [1]. The Global Criticality/Local Phase (GCLP) algorithm, firstly published in 1994 [2], has been integrated into Ptolemy in 1995 [3]. In the following years the authors enhanced this method to solve the *extended partitioning problem* [4], which incorporates the existence of several implementation *bins* for both hardware (HW) and software (SW). Due to its fine reputation being a fast technique, i.e. with a low complexity of $\mathcal{O}(|V|^2)$ in the number of processes $|V|$, while yielding reasonably good results compared to *Integer Linear Programming* [4], the Open Tool Integration Environment (OTIE) [5] has been enriched with a version of the GCLP algorithm. As our main focus lies on SoCs in the wireless domain with strict real-time constraints, the architecture abstraction is slightly different. It features a more sophisticated communication model to deliver precise timing results including bus traffic, different access times for read and write instructions and the distinction between local and shared memory units. The analysis and evaluation of the original algorithm disclosed several possibilities to save computation time and to improve quality. The contribution of this paper comprises a thorough analysis of the GCLP algorithm and the introduction of several modifications to increase the performance of this approach with respect to the solution quality, the computation time and the probability of valid results.
The rest of the paper is organised as follows. The next section sheds some light on related work in the field addressing combined partitioning/scheduling techniques. Section 3 illustrates the basic principles of system partitioning and gives an overview of the GCLP algorithm. It is followed by a detailed description of the new architecture model, the applied modifications in Sec. 4 and results for every single modification. Suitable combinations of the proposed modifications are compared to the original GCLP algorithm in Sec. 5. The work is concluded and perspectives to future work are given in Sec. 6 .

2. RELATED WORK

Heuristic approaches dominate the field of partitioning algorithms, since partitioning is known to be an NP-hard problem in most formulations [6]. Genetic algorithms have been extensively used [7, 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, which features a very low algorithmic complexity. With respect to combined partitioning/scheduling approaches, the work in [14, 15] has to be mentioned. The approaches in [16, 8] 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 [12, 9], which might be reconfigurable [14], to a limited set of several concurrently running HW units combined with a general-purpose processor [17, 18].

3. SYSTEM PARTITIONING WITH GCLP

This section covers the fundamentals of system partitioning and the main mechanisms of the GCLP algorithm. 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 [19, 20, 4].

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 a hard optimisation problem [21], in many formulations even NP-hard [6, 2]. The system functionality is typically abstracted into a graph $G = (V, E)$ representation. In Fig. 1a, six vertices $V = \{a, .., f\}$ are depicted which are connected by six edges $E = \{e_1, .., e_6\}$. 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 [4, 17, 16, 14] decide for medium sized vertices that cover the functionality of FIRs, IDCTs, shellsort algorithm or similar procedures. Every vertex has

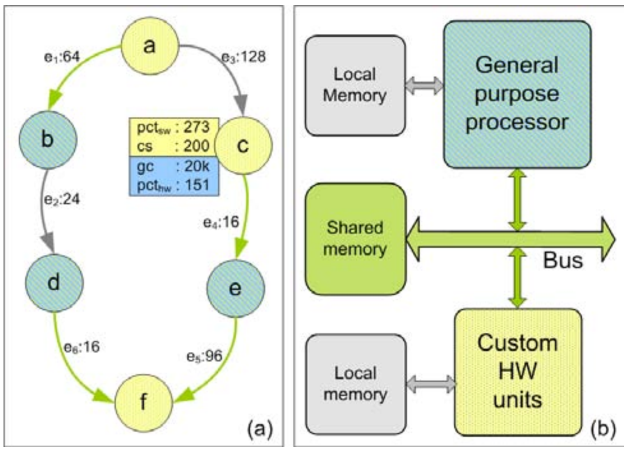
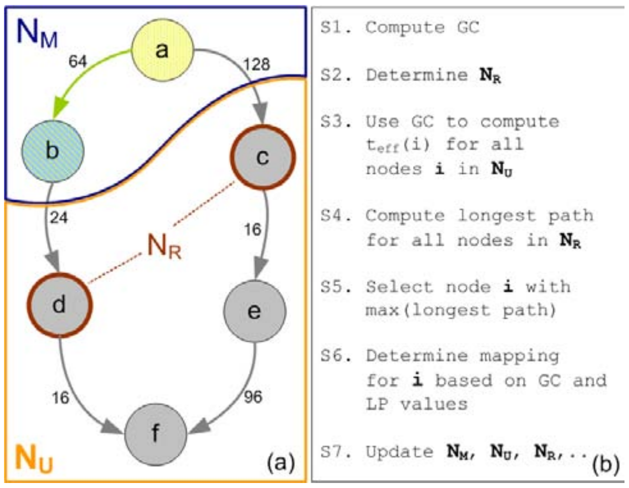


Figure 1: *(a) Process graph, annotated with characteristic values. (b) Typical platform model.*

been annotated with characteristic values, that, in the case of the GCLP algorithm for the binary (SW, HW) partitioning problem, build a quadruple: (*process computation time (pct_{sw}) and code size (cs) for SW, process computation time (pct_{hw}) and area in gates (gc) for HW*). The edges are annotated with the number of data samples (bytes) transmitted per invocation of one process. The mapping of the task graph to the given architecture in Fig. 1b is performed by the GCLP algorithm with the objective to meet constraints for time, 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. A model for HW to SW communication via shared memory is provided, whereas HW to HW and SW to SW communication is neglected. The following paragraphs present a short discussion of the basic concepts of the GCLP approach. For complete detail, please refer to the author's dissertation [3].

Essentially this algorithm is a greedy approach, which visits every vertex exactly once, and decides where to map it based on two different values: the *Global Criticality* (GC) measure and the *Local Phase* (LP) measure. The GC value is a *global* look-ahead measure that estimates whether time, code size or area is most critical at the current stage of the algorithm and then decides which of these targets shall be minimised. The LP value is calculated for every single process before the main algorithm starts and is based on intrinsic properties that represent the individual mapping preferences of this process. For instance, when a specific process prefers an implementation in SW, because of its very large bit level instruction mix, the LP value reflects this preference, or when a process stands out by its extraordinary HW size and a rather small SW execution time, then LP value takes this into account. By the superposition of the global GC value and the local LP value the greediness of the approach is moderated and a balanced mapping, which meets all constraints, shall be ensured.

In Fig. 2a the process graph is depicted and in Fig. 2b



```
S1. Compute GC
S2. Determine $N_R$
S3. Use GC to compute
    $t_{eff}(i)$ for all
    nodes i in $N_U$
S4. Compute longest path
    for all nodes in $N_R$
S5. Select node i with
    max(longest path)
S6. Determine mapping
    for i based on GC and
    LP values
S7. Update $N_M$, $N_U$, $N_R$, ..
```

Figure 2: *(a) Process graph at a distinct stage of the GCLP algorithm. (b) Pseudo code for a single GCLP iteration.*

pseudo code of one GCLP iteration is listed. The upper two vertices have been already **m**apped ($N_M = \{a, b\}$), all others are still **u**nmapped ($N_U = \{c, d, e, f\}$), of which two are **r**eady ($N_R = \{c, d\}$) to be mapped next. In step S1 the current GC value is calculated. Within S1 a provisional yet *complete* mapping is performed such that the time constraint is surely met. The GC value is then calculated based on this preliminary mapping and is normalised to lie in the interval $[0,1]$ ($0 \triangleq$ lowest time criticality, $1 \triangleq$ highest time criticality). In step S2 the ready processes $N_R = \{c,d\}$ are determined. The steps S3 and S4 shall decide which of both vertices c,d will be mapped next. In step S3, an *effective* execution time $t_{eff} = \mathrm{GC}pct_{hw} + (1-\mathrm{GC})pct_{sw}$ is assigned to all yet unmapped vertices. In step S4, t_{eff} serves as the base for a longest path search from every vertex in N_R to the exit process f. In step S5, the vertex with the maximum longest path value is selected to be mapped next. In step S6 the final mapping of this vertex is performed based on the superposition of the *global* GC value and the *local* LP value. In step S7, all sets, lists and intermediate values are updated. These seven steps are repeated until all vertices have been finally mapped ($N_U = \emptyset$).

The algorithmic complexity is claimed to be $\mathcal{O}(|V|^2)$, $|V|$ is the number of vertices. The step S1 is the most dominant part, since several provisional mappings are produced and a complete schedule has to be generated for every single one of them, until the deadline is met. An underlying list scheduling technique, *earliest task first*, ensures a *re*-scheduling time, which is linear in the number of vertices and edges $\mathcal{O}(|V|+|E|)$. For *sparse* graphs ($|V| \sim |E|$) this is usually simplified to $\mathcal{O}(|V|)$. The second dominant part are the steps S3 and S4: S3 is a list operation on an ever decreasing list of length $|V|$, thus the complexity is approximated with $\mathcal{O}(|V|/2) = \mathcal{O}(|V|)$, and S4 is a longest path search over the subgraph of the N_U node set. It can be implemented with linear complexity $\mathcal{O}(|V|+|E|)$, set the graph is directed and acyclic (DAG). S7 comprises several list updates that are linear as well. Since the leading terms are all linear and these seven steps are performed for all $|V|$ vertices in the graph, the result grows asymptotically with squared complexity: $\mathcal{O}(|V|^2)$.

4. IMPROVEMENTS

The section discusses in detail the improvements applied to underlying platform model and the GCLP algorithm itself. In the following the algorithm is evaluated based on some characteristic values: the computational run time Θ, on a PC (AMD Athlon 64 3000+, 1.8GHz Processor) measured in seconds, the quality of the obtained solution, the cost value Ω_P, for the best partitioning solution P:

$$\Omega_P = \alpha\frac{T_P}{T_{limit}} + \beta\frac{A_P}{A_{limit}} + \gamma\frac{S_P}{S_{limit}}. \qquad (1)$$

The makespan of the graph for P is T_P, which must not exceed T_{limit}. The sum over the area of all processes mapped to HW is A_P, which must not exceed A_{limit}. The sum over the code sizes of all processes mapped to SW is S_P, which must not exceed S_{limit}. With the weight factors α, β, and γ the designer can set individual priorities. If not stated otherwise, these factors are set to 1.0. The boolean *validity* V_P of an obtained partitioning P is given by the boolean expression: $V_P = (T \le T_{limit}) \wedge (A_P \le A_{limit}) \wedge (S_P \le S_{limit})$. A last characteristic value is the validity percentage $\Upsilon = N_{valid}/N$, which 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.
Among system partitioning techniques this approach stands out because of its low algorithmic complexity ($\mathcal{O}(|V|^2)$). The key aspect of its design is to find solutions that meet the constraints as fast as possible rather than traversing the vast search space in a time-consuming manner. Thus, the objective for all the following considerations focuses on either the improvement of the solution quality Ω_P without affecting the run time Θ and the validity percentage Υ, or on a substantial reduction of Θ without affecting Υ.
A multitude of graphs have been generated according to the same rules as described in the original work [3]. The sets are ordered by the size of the contained graphs, measured by the number of vertices $|V|$. Each set contains 180 different graphs of the same size.
The constraints are specified by three ratios R_T, R_A, R_S obtained by the following equations:

$$R_T = \frac{T_{limit} - T_{min}}{T_{total} - T_{min}}, R_A = \frac{A_{limit}}{A_{total}}, R_S = \frac{S_{limit}}{S_{total}}, \qquad (2)$$

The totalised values for area A_{total}, code size S_{total}, and execution time T_{total} are simply the sum over the gate counts gc, code sizes cs, and SW execution times pct_{sw} (plus communication) of all processes. The computation of T_{min} is obtained by scheduling the graph under the assumption of a pure HW implementation featuring a full parallelism, i.e. unlimited HW resources. Therefore, a constraint is rather strict, when the allowed resource limit is small in comparison to the resource demands that are present in the graph. For instance, the totalised gate count A_{total} of all processes in the graph is $100k$ gates, if $A_{limit} = 20k$, then $R_A = 0.2$, which is rather strict, as in average only every fifth process may be mapped to HW at all. If not stated otherwise, medium constraints ($R_T = R_A = R_S = 0.5$) are set as targets.

4.1 Extended Platform Model

As mentioned before the communication model of the original GCLP algorithm completely neglects HW-HW and SW-SW data transfers. Apparently the major load and store procedures for data and instructions that occur at the beginning and end of every process do affect the execution time of the system. The abstracted HW processor does not properly reflect the realities that we face in the wireless domain. The inspiration for the architecture model in this work originates from an industry-designed UMTS baseband receiver chip [22, 18]. Its abstraction (see Figure 3) has been developed to provide a maximum degree of generality while being along the lines of the industry-designed SoCs in use. It consists of at least one (or more) DSP handling the control oriented functions, for instance an ARM [23] for the signalling part (and/or a StarCore [24] for the multimedia part), several hardware accelerating units (ASICs), for the data oriented and computation intensive signal processing, one system bus to a shared RAM for mixed resource communication, and optionally direct I/O to the periphery, i.e. the antenna subsystem. To capture the communication times precisely the designer is allowed to model load and store times for the interprocess data transfers. Table 1 lists the access times for reading and writing bits via the different resources of the platform in Figure 3.
All memory and bus resources are properly scheduled during the HW/SW partitioning process. Collisions are solved via the HU level list scheduling technique [25]. The priority levels that are required for every single process have to be computed beforehand based on the unpartitioned task graph.

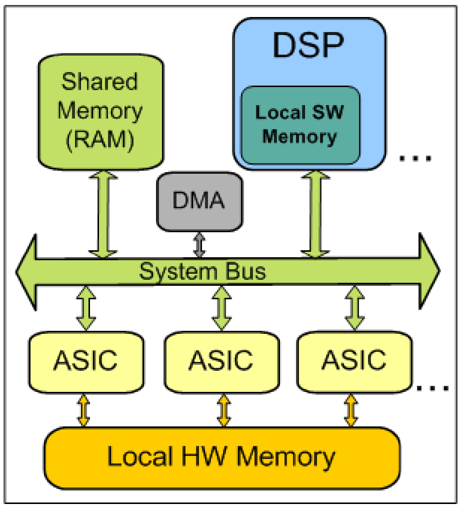


Figure 3: Extended SoC platform abstraction with dedicated *load* and *store* times for all communication resources.

Communication	read (bits/cycle)	write (bits/cycle)
Local SW memory	128	256
Local HW memory	64	128
Shared system bus	256	512
Direct I/O	1024	1024

Table 1: Maximum throughput for read/write accesses to the communication resources.

4.2 Modification 1 - Revision of S3 and S4

Consider the steps S3 and S4 in the listing in Fig. 2b. Note, that their single purpose is the decision *which* process is going to be mapped next, neither *where* it is going to be mapped, nor *when* exactly it will be scheduled when *all* processes have been finally mapped. For all the graph sets, a positive impact on the solution quality by these two steps could not be observed. A comparison to a random selection of the process from N_R, which should be mapped next, did not show any significant difference, as Tables 2 and 3 indicate. The reason for this result is two-fold: the calculation of the longest paths in S4 is based on *effective* execution times. The longest path search yields correct values for all vertices in N_R, if and only if $\text{GC} = 1$, or in other words in case of a complete HW solution, given the HW processor allows for **concurrent** execution of tasks. For a complete SW solution, the longest path calculation loses its relation to the graph completely, since the SW processor is a **sequential** device, and all processes have to run on it consecutively anyway. So for small GC values this calculation does not have significance, and for balanced GC values, the execution times are averaged between pct_{hw} and pct_{sw} and lack precision due to this averaging. Only for large GC values S4 delivers approximately correct results, which is not enough to compensate the imbalance of this mechanism.
To overcome this malfunction we propose two modifications, M1a or M1b:

- M1a: Omit the steps S3 and S4 completely to save run time of about 15%. That is only of interest for huge graphs ($|V| > 500$), in which the run time for each graph becomes a matter of many seconds instead of milliseconds.
- M1b: Calculate the longest path searches for all vertices in N_R based on the provisional partitioning just generated in step S1. Recall, that step S1 comprises a full partitioning and scheduling to compute the current GC value and thus represents a precise snapshot of the present partitioning situation: all processes apply either their correct pct_{sw} or pct_{hw} instead of a mixture of both and a full schedule exists, hence, the longest path search in S4 returns correct values to determine the vertex in N_R that currently lies on the critical path. S3 can be saved here as well.

Graphs ($\|V\|$)	Cumulated run time (Θ_{cum})			Cumulated cost (Ω_{cum})		
	GCLP	M1a	M1b	GCLP	M1a	M1b
20	1.3s	1.2s	1.3s	292.1	293.8	290.4
50	8.2s	7.3s	8.1s	287.3	286.4	282.7
100	47.5s	42.6s	46.0s	281.6	281.5	276.9
200	627.8s	542.0s	619.1s	278.6	279.0	273.2

Table 2: *Impact on run time and cost of proposed modifications M1a and M1b compared with the original GCLP algorithm.*

Table 2 shows the impact for all graph sets on run time and cost. M1a saves about 15% run time without any degradation of the obtained solutions. Modification M1b improves the result quality by about 1.5% to 2% in cost, *and* reduces the run time, *and* features an almost 3% higher Υ, as listed in Tab. 3.

Graphs ($\|V\|$)	Υ(%)		
	GCLP	M1a	M1b
20	74.4	75.0	76.6
50	86.1	86.6	90.0
100	90.0	90.0	92.2
200	90.0	90.5	92.2

Table 3: *Impact on the percentage of valid solutions* Υ.

4.3 Modification 2 - Initial Solution

Another substantial gain in performance is possible by a more sophisticated choice of the initial solution. Although the preparation phase of GCLP comprises the individual characterisation of processes with respect to their preferred implementation type, GCLP assumes a complete SW solution as starting point. Neither the constraints given by the designer nor the just calculated LP values affect this assumption in any manner. A strong potential to enhance the quality of the final result without increasing the run time can be put forth. The initial configuration for GCLP is a graph, in

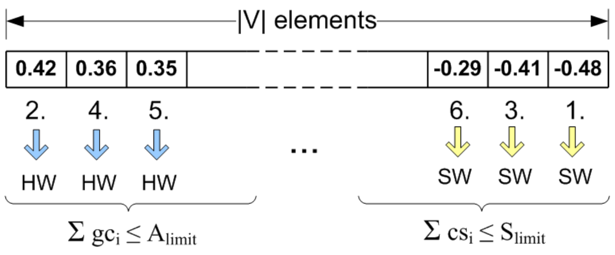


Figure 4: *Modification 2 (M2): Constructing the initial solution.*

which every vertex has an LP value in $[-0.5, 0.5]$ indicating whether it is more suited for a SW (-0.5) or a HW (0.5) implementation. The generation of these values is described in

detail in the publications [3, 4], due to limited space it has to be omitted here. A simple and fast strategy to construct a better initial solution is to build an ordered list of these individual values, which can be achieved very efficiently (on average in $\mathcal{O}(|V|\log|V|)$ with the quicksort algorithm).
Now we process this list alternating from both ends, depending on the absolute value of the contained measure, as depicted in Fig. 4. We proceed as long as the initially mapped processes do not reach the area limit A_{limit} for those mapped to HW or the code size limit S_{limit} for those mapped to SW. The remaining processes in the middle of this list are flagged to be considered preferentially in step S1 of the GCLP algorithm. The complexity of this operation is $\mathcal{O}(|V|)$. The computational overhead is smaller than 0.3% and was only observable during the simulations for the largest graphs ($|V| = 200$) cumulated over 180 graphs. Table 4 contains the

Graphs	Cumulated cost (Ω_{cum})		Υ (%)	
($\|V\|$)	GCLP	M2	GCLP	M2
20	292.1	290.2	74.4	76.6
50	287.3	283.0	86.1	88.8
100	281.6	276.9	90.0	91.1
200	278.6	273.3	90.5	91.6

Table 4: *Impact on cost and validity percentage of M2.*

results obtained while applying this modification (M2) to the graph sets compared to the original algorithm. Another gain in cost and a higher yield in valid results can be achieved.

4.4 Modification 3 - Precocious Breaks

A third modification (M3) is the insertion of precocious breaks as soon as all constraints are met. Although the design of the GCLP algorithm focused on low run time, a mechanism to stop the algorithm as soon as possible is surprisingly not provided. As stated before, step S1 generates a full partitioning solution, even though being provisional, it makes perfect sense to evaluate this solution as well. The partitioning with the lowest cost seen is stored and when the constraints happen to be met, the algorithm stops. In the case of rather loose constraints the run time drops dramatically. When the constraints are rather strict, so that the original algorithm would finalise returning an *invalid* solution, the run time stays exactly the same, with a possibly better cost obtained by one of the provisional mappings. When the constraints are strict, but the original algorithm would finalise with a *valid* solution, the run time will drop very likely by at least a small margin. For a profound understanding of the last case, it is mandatory to demonstrate the functionality of S1 in detail.
As stated in Sec. 4.2, in S1 it is always assumed that all processes in N_U are implemented in SW. Then it moves tentatively processes to HW until the time constraint is met. Of course this mechanism is sensitive to the chosen order in which the processes in N_U are tentatively moved. The GCLP designers proposed an priority list for the processes ordered by their best *gain* in time measured by the quotient pct_{sw}/pct_{hw}. A large *gain* means that its mapping from SW to HW results very likely in a large execution time reduction. Consider a situation, which adheres to the mentioned case: a *valid* solution exists, that would be found by the original algorithm and rather strict constraints prevented a precocious break up to the current stage of the algorithm. In Fig. 5 the tail of a graph is depicted with the exit vertex z. The preceding iteration, in which process y was finally mapped, did not

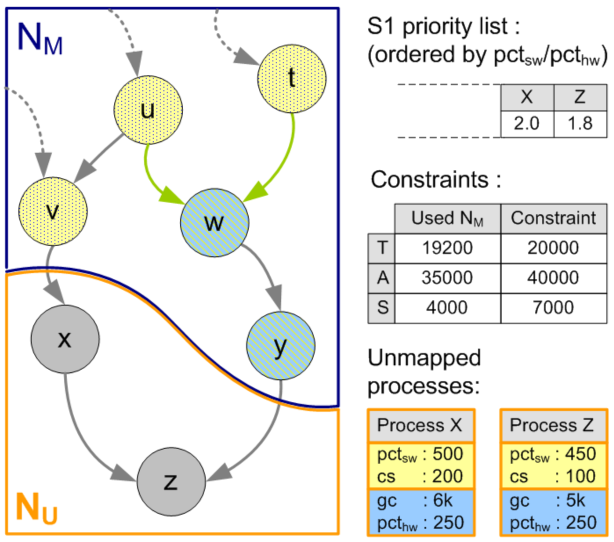


Figure 5: *Modification 3(M3): Precocious breaks.*

break precociously, i.e. not all constraints had been fulfilled in S1 of the last iteration. Since S1 ensures a provisional partitioning, in which T_{limit} is met, only A_{limit} and/or S_{limit} could have been exceeded. But this is only possible when the order of the priority list, that guides the tentative mapping, in S1 does *not* cause a valid mapping. On the top right of Fig. 5 the entries for x and z in the priority list are shown. Hence, S1 does always map x to HW at first, detects that T_{limit} is met and thus leaves z in SW. In this example A_{limit} is then exceeded by this combination ($35000 + 6000 \geq 40000$), so a precocious break is not possible. The following final mapping of x chooses a SW implementation, since A_{limit} is exceeded whereas T_{limit} is met and proceeds the very last process z in the graph.
This scenario demonstrates the only case in which the modified version is not capable of finishing at least a short time earlier than the original algorithm. In *all* other scenarios, when the tail of the priority list matches a *valid* partitioning solution, a precocious break will occur. Table 5 lists the impacts of this last modification on the run time for loose, medium, and strict constraints. The leftmost column

Constraints	Cumulated run times Θ_{cum}			
	$\|V\| = 20$		$\|V\| = 50$	
(R_T, R_A, R_S)	GCLP	M3	GCLP	M3
(0.4, 0.4, 0.4)	1.3s	1.3s	8.2s	8.1s
(0.5, 0.5, 0.5)	1.3s	1.2s	8.2s	7.9s
(0.7, 0.7, 0.7)	1.3s	1.2s	8.1s	7.2s
Constraints	$\|V\| = 100$		$\|V\| = 200$	
(R_T, R_A, R_S)	GCLP	M3	GCLP	M3
(0.4, 0.4, 0.4)	48.0s	45.4s	632.4s	548.3s
(0.5, 0.5, 0.5)	47.5s	43.2s	627.8s	525.0s
(0.7, 0.7, 0.7)	47.1s	39.8s	623.0s	498.6s

Table 5: *Effect of modification M3 on the run time.*

of Tab. 5 contains a set of constraint ratios (R_T, R_A, R_S). The run time improvement for large graphs and loose constraints is substantial with up to 21%. The validity percentage is even improved by about 0.5% for larger graphs ($|V| \geq 100$), as there are rare occasions, when a provisional mapping is detected to be valid and the modified algorithm ends precociously, whereas the original algorithm would yield an invalid result with one of the constraints narrowly missed.
It has to be mentioned that the third modification evidently

leads to a degradation of the quality for the *valid* partitioning solutions, as a precocious break is surely valid but will often have a higher cost than an algorithm with this option disabled. Whereas the quality of *invalid* solutions will increase, as the provisional mappings are considered additionally.

5. RESULTS FOR COMBINED MODIFICATIONS

Eventually, two promising combinations, C1 and C2, of the proposed modifications are build. Combination 1 incorporates M1a and M3 to obtain an algorithm with a substantially lower run time Θ, a slightly better validity percentage Υ, and minor degradations of the solutions cost Ω. Combination 2 incorporates M1b and M2 to obtain an algorithm, which concentrates on cost improvements and higher validity percentages without affecting the run time. The succeeding Tables 6-8 present a comparison with the original algorithm for all graph sets and different sets of constraints:

Table 6 lists the significant improvements of C1 concerning run time. Naturally, large graphs with rather loose constraints lead to a dramatic drop in computation time of up to 27%. Additionally C1 causes a measureable increase in the validity percentage of about 1%. These improvements are paid by a rise in cumulated cost Ω_{cum} of about 3-4%. The combination C2 is a more balanced improvement. The predominant part is the boost in validity percentage Υ, with about 4% most noticeable for strict constraints on smaller graphs. This performance is accompanied by a reduction of cost of up to 3%, while the run time even drops slightly.

Both combinations cover different areas of problem instances, while both prove to be better than the original algorithm in these areas. The first combination C1 is recommended for problem instances with very large graphs ($|V| \geq 200$) or a graph set containing many different graphs, for which valid results shall be produced, as its benefits lie predominantly in a run time reduction. The second combination C2 can simply replace the implementation of the original GCLP algorithm, as it yields better results in every aspect with the largest margin in increasing Υ.
Finally it has to be clarified that the GCLP approach was not designed and is not capable to compete with time-consuming approaches based on genetic algorithms, tabu search, simulated annealing or even integer linear programming, when the aim is to find a near-optimal solution. The run time of these approaches is $10^3 - 10^4$ times higher [17, 4], while tens of thousands of solutions are generated and a cost reduction of up to 15% is observed.

6. CONCLUSIONS

In this work the GCLP algorithm for the solution of the binary HW/SW partitioning problem has been thoroughly analysed and several modifications to increase its performance have been introduced. Depending on the problem instances and the designer's intentions two versions of GCLP advancements are presented, either of which yielding significantly better results than the original algorithm with the focus set on different problem instances. The introduction of a more sophisticated platform model increases the reliability of the results further. Precise static schedules can be generated for all resources in the design.

Future work will concentrate on low complexity techniques exploiting the inherent parallelism in the graph structure. Since more than one HW implementation alternative may exist for a single process depending on the pipelining, the loop unrolling factor or allowed register usage, a more complex scenario has to be embraced by the partitioning strategy than in the case of binary decisions between HW and SW. Analogously control-oriented functions may cover a range of different execution times based on their current control flow. In a further step these execution time profiles shall be integrated in the final partitioning technique.

	Cumulated run times Θ_{cum}					
Constraints	$\|V\| = 20$			$\|V\| = 50$		
(R_T, R_A, R_S)	GCLP	C1	C2	GCLP	C1	C2
(0.4, 0.4, 0.4)	1.3s	1.2s	1.3s	8.2s	8.0s	8.1s
(0.5, 0.5, 0.5)	1.3s	1.2s	1.3s	8.2s	7.7s	8.1s
(0.7, 0.7, 0.7)	1.3s	1.1s	1.2s	8.1s	6.8s	8.0s
Constraints	$\|V\| = 100$			$\|V\| = 200$		
(R_T, R_A, R_S)	GCLP	C1	C2	GCLP	C1	C2
(0.4, 0.4, 0.4)	48.0s	41.4s	47.2s	632.4s	529.5s	626.1s
(0.5, 0.5, 0.5)	47.5s	40.9s	46.9s	627.8s	503.8s	625.7s
(0.7, 0.7, 0.7)	47.1s	37.3s	46.1s	623.0s	458.3s	621.2s

Table 6: *Impact of combined modifications C1 (M1a, M3) and C2 (M1b, M2) on the cumulated run time Θ_{cum}.*

	Cumulated cost Ω_{cum}					
Constraints	$\|V\| = 20$			$\|V\| = 50$		
(R_T, R_A, R_S)	GCLP	C1	C2	GCLP	C1	C2
(0.4, 0.4, 0.4)	321.2	330.4	317.4	314.9	323.4	309.1
(0.5, 0.5, 0.5)	292.1	302.8	287.0	287.3	298.5	282.2
(0.7, 0.7, 0.7)	250.6	266.9	245.2	244.0	256.0	239.2
Constraints	$\|V\| = 100$			$\|V\| = 200$		
(R_T, R_A, R_S)	GCLP	C1	C2	GCLP	C1	C2
(0.4, 0.4, 0.4)	308.5	319.4	301.9	305.4	312.3	297.7
(0.5, 0.5, 0.5)	281.6	292.7	275.4	278.6	291.0	271.1
(0.7, 0.7, 0.7)	238.7	250.2	233.3	231.4	243.9	222.8

Table 7: *Impact of combined modifications C1 (M1a, M3) and C2 (M1b, M2) on the cumulated cost Ω_{cum}.*

	Validity percentage Υ					
Constraints	$\|V\| = 20$			$\|V\| = 50$		
(R_T, R_A, R_S)	GCLP	C1	C2	GCLP	C1	C2
(0.4, 0.4, 0.4)	58.8	59.4	62.7	68.8	69.4	69.4
(0.5, 0.5, 0.5)	73.8	74.4	77.7	86.1	86.6	88.8
(0.7, 0.7, 0.7)	97.9	97.9	98.4	98.8	98.8	100.0
Constraints	$\|V\| = 100$			$\|V\| = 200$		
(R_T, R_A, R_S)	GCLP	C1	C2	GCLP	C1	C2
(0.4, 0.4, 0.4)	65.0	66.1	68.8	71.6	72.2	73.3
(0.5, 0.5, 0.5)	90.0	90.5	92.2	90.5	91.5	92.2
(0.7, 0.7, 0.7)	100.0	100.0	100.0	100.0	100.0	100.0

Table 8: *Impact of combined modifications C1 (M1a, M3) and C2 (M1b, M2) on the validity percentage Υ.*

REFERENCES

[1] E.A. Lee. Overview of the Ptolemy Project. Technical report, University of Berkeley, March 2001. http://ptolemy.eecs.berkeley.edu.

[2] A. Kalavade and E. A. Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *CODES '94: Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 42–48, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[3] A. Kalavade. *System-level codesign of mixed hardware-software systems*. PhD thesis, University of California, Berkeley, CA, USA, 1995.

[4] A. Kalavade and E.A. Lee. The extended partitioning problem: hardware/software mapping, scheduling,

and implementation-bin selection. *Readings in hardware/software co-design*, pages 293–312, 2002.

[5] P. Belanović, B. Knerr, M. Holzer, G. Sauzon, and M. Rupp. A consistent design methodology for wireless embedded systems. *EURASIP Journal on Applied Signal Processing*, Vol. 2005(16):2598–2612, 2005.

[6] P. Arató, Z. Á. Mann, and A. Orbán. Algorithmic aspects of hardware/software partitioning. *ACM Trans. Des. Autom. Electron. Syst.*, 10(1):136–156, 2005.

[7] V. Srinivasan, S. Radhakrishnan, and R. Vemuri. Hardware/software partitioning with integrated hardware design space exploration. In *DATE '98: Proceedings of the conference on Design, automation and test in Europe*, pages 28–35, Washington, DC, USA, 1998. IEEE Computer Society.

[8] B. Mei, P. Schaumont, and S. Vernalde. A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems. In *Proceedings of ProRISC*, 2000.

[9] J. Henkel and R. Ernst. An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(2):273–290, 2001.

[10] M. L. Lopez-Vallejo and J. Grajal and J. C. Lopez. Constraint-driven system partitioning. In *Proceedings of Design, Automation & Test in Europe (DATE)*, pages 411–416, 2000.

[11] P. Eles and Z. Peng and K. Kuchcinski and A. Doboli. System level hardware/software partitioning based on simulated annealing and tabu search. *Design Automation for Embedded Systems*, 2:5–32, 1997.

[12] J. Grode, P. V. Knudsen, and J. Madsen. Hardware resource allocation for hardware/software partitioning in the lycos system. In *Proceedings of Design, Automation & Test in Europe (DATE)*, pages 22–27, Washington, DC, USA, 1998. IEEE Computer Society.

[13] R.K. Gupta and G. De Micheli. Hardware-software cosynthesis for digital systems. *Readings in hardware/software co-design*, pages 5–17, 2002.

[14] K.S. Chatha and R. Vemuri. Magellan: multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs. In *Proceedings of the ninth international symposium on Hardware/software codesign (CODES)*, pages 42–47, New York, NY, USA, 2001. ACM Press.

[15] M. Lopez-Vallejo and J.C. Lopez. On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Trans. Des. Autom. Electron. Syst.*, 8(3):269–297, 2003.

[16] R.P. Dick and N.K. Jha. Mogac: a multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 522–529, Washington, DC, USA, 1997. IEEE Computer Society.

[17] T. Wiangtong, P.Y.K. Cheung, and W. Luk. Comparing three heuristic search methods for functional partitioning in hardware-software codesign. *Design Automation for Embedded Systems*, 6(4):425–449, Sept 2002.

[18] Bastian Knerr, Martin Holzer, and Markus Rupp. A fast rescheduling heuristic of sdf graphs for hw/sw partitioning algorithms. In *Proc. of IEEE Conference on Communication System, Software and Middleware 2006*, New Delhi, India, January 2006.

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

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

[21] J. Hromkovič. *Algorithmics for hard problems*. Springer-Verlag, Inc., New York, NY, USA, 2nd edition, 2004.

[22] B. Knerr, M. Holzer and M. Rupp. Fast rescheduling of multi-rate systems for hw/sw partitioning algorithms. In *Proc. of Thirty-Ninth Annual Asilomar Conference on Signals, Systems, and Computers*, Monterey, CA, USA, October 2005.

[23] ARM9 Core Family. Technical report. http://www.arm.com/products/CPUs/families/ARM9 Family.html.

[24] StarCore SC1000 Family. Technical report. http://www.starcore-dsp.com/products/sc1000/index.shtml.

[25] T. C. Hu. Parallel Sequencing and Assembly Line Problems. Technical Report 6, Operations Research, 1961.