# Fast Rescheduling of Multi-Rate Systems for HW/SW Partitioning Algorithms

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

Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Vienna, Austria
{bknerr, mholzer, mrupp}@nt.tuwien.ac.at

## Abstract

*In modern designs for heterogeneous systems with their extreme requirements on power consumption, execution time, silicon area and time-to-market, the HW/SW partitioning problem belongs to the most challenging ones. Usually its formulation, based on task or process graphs with complex communication models, is intractable. Moreover most partitioning problems embed another NP-hard problem in its core: a huge number of valid schedules exist for a single partitioning solution. Powerful heuristics for the partitioning problem rely on list scheduling techniques to solve this scheduling problem. This paper is based on a rescheduling algorithm that performs better than popular list scheduling techniques and still preserves linear complexity by reusing former schedules. A sophisticated communication model is introduced and the rescheduling algorithm is modified to serve multi-core architectures with linear runtime.*

**Keywords:** Task Scheduling, HW/SW Partitioning, Multi-Rate Systems, Fast Rescheduling, Multi-Core Platforms

## 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 lead 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 the following section.

In most HW/SW partitioning approaches heuristics are used to traverse the search space, see Sec. 2. The embedded scheduling problem is *always* solved by list scheduling algorithms and/or estimations about the expected system runtime. In our earlier work we have shown that a better scheduling technique can be incorporated in many HW/SW partitioning algorithms permitting more reliable results of the partitioning process [11]. This work enriches the proposed rescheduling technique with a more realistic communication model and provides the extension to multi-core architectures.

The rest of the paper is organised as follows: after the related work section the common system abstraction into SDF graphs is briefly introduced. Afterwards the HW platform model, which the system should be mapped on, is described. Sec. 5 discusses the complexity that partitioning engines have to deal with. In Sec. 6 the fundamental idea of the proposed rescheduling algorithm and the extension to multi-core platforms are presented. In Sec. 7 results, obtained by its application to typical task graph sets, are presented. Sec. 8 concludes the paper.

## 2 Related Work

Heuristic approaches dominate the field of partitioning algorithms, since partitioning is known to be an NP-hard problem in most formulations [1]. Genetic algorithms [14] have been extensively used as well as simulated annealing [7]. To a smaller degree tabu search [3] and greedy algorithms [5] have also been applied. Other research groups developed custom heuristics such as the GCLP algorithm in [10] or the early work in [6].

Formulations of the inherent multi-processor scheduling problem have been shown to be NP-complete [4, 12]. The

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

vital importance of this task caught permanent attention of research groups over the last 20 years. To name only a few with low complexity: Hu's level based scheduling [8], the dynamic level scheduling approach of Sih [17], the edge-zeroing algorithm [16], which focuses on communication costs, and the mobility directed algorithm [20].

With respect to combined partitioning/scheduling approaches, the work in [2] has to be mentioned. The approach in [14] also adds communication events to links between hardware and software units. The architecture model varies from having a single software and a single hardware unit [5, 7], which might be reconfigurable [2], to a limited set of different hardware units in combination with a general-purpose processor.

The work of Theerayod et al. [18] is mentionable due to the detailed architecture and communication model and the comprehensive evaluation of three different partitioning heuristics: a simulated annealing approach, a genetic algorithm and a sophisticated tabu search implementation.

Approaches showing the *one move* step behaviour of the algorithm through the search space can be found in [3,7,18, 19].

## 3 Signal Processing Systems as SDF Graphs

The graph representation of the system is generally given in the form of a task graph, which is usually assumed to be a directed acyclic graph (DAG) describing the dependencies between the components of the system. We based our work on the a graph representation for multi-rate systems, also known as synchronous data flow graphs [13]. This representation accomplished the backbone of reknowned signal processing work suites, e.g. Ptolemy or CoWare SPW 4. In

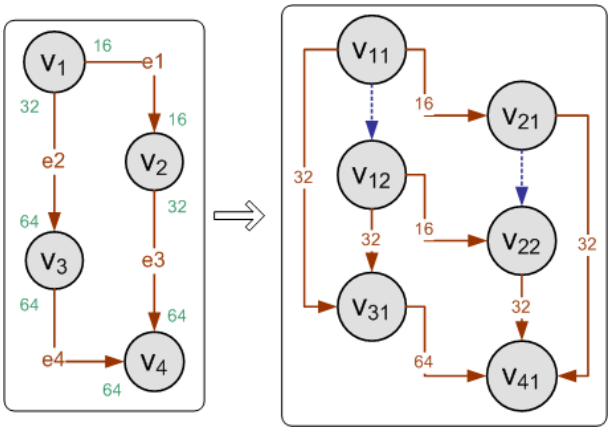


**Figure 1.** *Simple SDFG (left) and the decomposition into its SAG (right).*

Fig. 1 a simple example of an SDF graph $G = (V, E)$ is depicted on the left, showing four vertices $V = \{v_1, .., v_4\}$. These are connected by four edges $E = \{e_1, .., e_4\}$. The numbers on the tail of each edge represent the number of bits produced per invocation. The numbers on the head of each edge indicate the number of bits consumed per invocation. On the right the decomposition of the SDF graph has been performed. In the single activation graph (SAG) the input/output rate dependencies have been solved and every *process invocation* is transformed into one vertex. The vertices $v_1$ and $v_2$ are doubled according to their distinct invocations that result from the data rate analysis. The solid edges indicate precedence as well as data transfers from one vertex to another, whereas the dashed edges just indicate precedence. The data rates at the edges, i.e. interprocess communication, have been omitted in this figure.

The decomposition of the system under investigation into an SDF graph depends heavily on the chosen level of granularity. We assume a vertex to be a sequential code block, further on also referred to as process, like a distinct matched filter function (e.g. DCT, FIR), or a sorting algorithm (e.g. shellsort), in opposition to be a single operational unit as e.g. a MAC or ALU. This granularity enables the identification of whole code blocks that may be implemented as a dedicated HW accelerator (ASIC) or run as SW on a DSP.

## 4 Model of the Hardware Platform

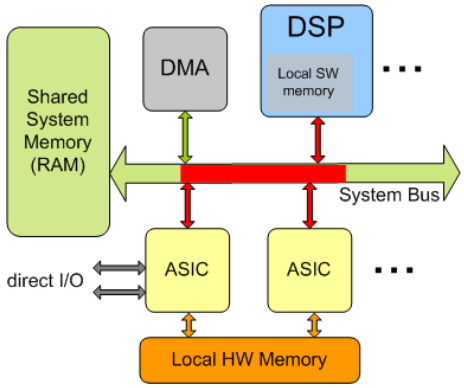


**Figure 2.** *General model of the target architecture.*

The target architecture model (see Fig. 2) has been chosen to provide a maximum degree of generality while being along the lines of the industry-designed SoCs in use. The platform has been abstracted from a UMTS baseband receiver chip: it consists in its least complex layout of one DSP, handling the control oriented functions, several HW accelerating units (ASICs), for the data oriented and computation intensive signal processing, a local hardware memory for inter-ASIC communication, one system bus to a shared RAM for mixed resource communication, and optionally direct I/O to peripheral subsystems. Table 1 lists the access times for reading and writing bits via the different resources:

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

**Table 1.** *Delay for interresource communication of the hardware platform.*

## 5 HW/SW Partitioning Process

HW/SW partitioning is supposed to identify the mapping of the vertices either to a SW module running on the DSP or to an ASIC being interfaced by the system bus, while meeting timing constraints, and minimise area and power consumption. A certain partitioning solution is characterised by a unique mapping of the four processes in Fig. 1 to either HW or SW, which results in a search space of $2^{|V|}$, with $|V| = 4$ vertices. Now assume the vertices $v_1$ and $v_2$ in Fig. 1 are SW modules being executed sequentially on the DSP, whereas $v_3$ and $v_4$ are HW modules, enabling full parallelism. For this single partitioning solution two valid DSP schedules are depicted in Fig. 3. Schedule 1 allows for a better exploitation of the inherent parallelism, which results in a shorter possible runtime. The read areas highlight the locations of beneficial permutations that lead to more parallelism. This simple example shows also

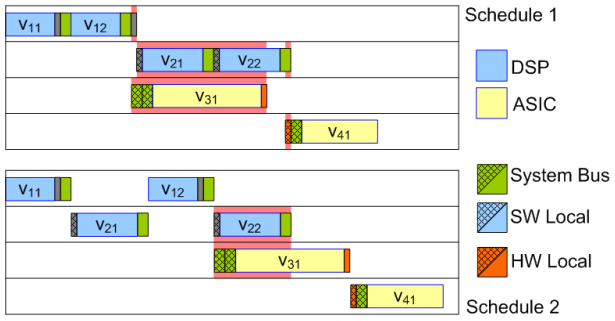


**Figure 3.** *Two valid schedules for a single partitioning solution with different runtimes.*

the complexity brought in by the detailed description of the read (dashed) and write (solid) accesses to the memory resources. All the approaches in the field of HW/SW partitioning solve the embedded scheduling by fast list scheduling techniques, often combined with estimations about the expected system execution time [7, 18, 19]. The main reason lies in the intractability of the resource allocation problem and its NP-completeness in most problem formulations [4, 12]. In this work we concentrate on partitioning algorithms that perform direct neighborhood searches while traversing the search space of partitioning solutions. In other words, algorithms like tabu search, simulated annealing or Kernighan-Lin adaptations generate new partitioning solutions by changing the implementation type of a *single* process at a time and evaluate the newly generated solution. This *one-move* behaviour allows for a reuse of the former schedule. It is possible to derive better scheduling decisions than those based on popular list scheduling techniques while still preserving linear complexity in the rescheduling of the new partitioning solution.

## 6 Rescheduling of SDF Graphs and Extension to Multi-Core Platforms

This chapter recalls briefly the main idea behind the fast rescheduling algorithm: the local exploitation of parallelism (LEP). Assume the following situation: a partitioning engine traverses the search space by moving one process at a time and it sees the schedule possibilities in Fig. 4. Process *X* was already on the DSP (blue), process *Y* is chosen to be moved next to the DSP (blue-yellow dashed) and the remaining processes are either HW (yellow) or SW (blue). The current rescheduling decision can always be reduced to a choice between two possible orderings as long as we move incrementally through the partitioning search space. In Fig. 4*a* process *X* is scheduled before *Y*, which is pushed back, and vice versa in Fig. 4*b*. Intuitively the better ordering can be identified by *getting more work done* within a limited time range $T_L$, here $T_L = max(X.start, Y.start) + X.time + Y.time$. This evaluation can be done as a restricted subgraph search for both orderings. For detailed information about the properties of the graph (*k-locality*, *density*, *degree of parallelism*) that are necessary to preserve linear complexity refer to the previous work in [11].

The extension towards multi-core architectures was inspired by an industry design of a UMTS baseband receiver chip that has been composed of two DSPs besides from the HW accelerators (ASICs): an ARM processor for the signalling part and a StarCore DSP (SC) for the multimedia part. Our goal is the extension of the general strategy of the LEP algorithm without analysing all possible permutations to preserve linear algorithmic complexity.

Assume the processes manage a list $L$ with tupels of their process computation times $pct$ on the respective processor in increasing order, e.g. process $i$ has $pct_{SC} = 20$ and $pct_{ARM} = 25$ results in a list $L_i = ((20, SC), (25, ARM))$. When the partitioning engine tries to move a process $i$ to SW, the extended LEP algorithm does as follows:

```
Foreach DSP in L_i {                                        //00
 if (NoCollision()) { MapTo(DSP); UpdSchedule();            //10
                      Return; }                             //20
}                                                           //30
CalculateT_L(firstDSPinL_i);                                //40
Foreach DSP in L_i {                                        //50
  // see Fig. 4                                             //60
  basic_LEP(); StoreBestResult(BestDSP, BestOrder);         //70
}                                                           //80
MapTo(BestDSP,BestOrder); UpdSchedule();                    //90
```

In lines 00-30 the algorithm simply checks for any collisions and maps the process immediately when it finds a free

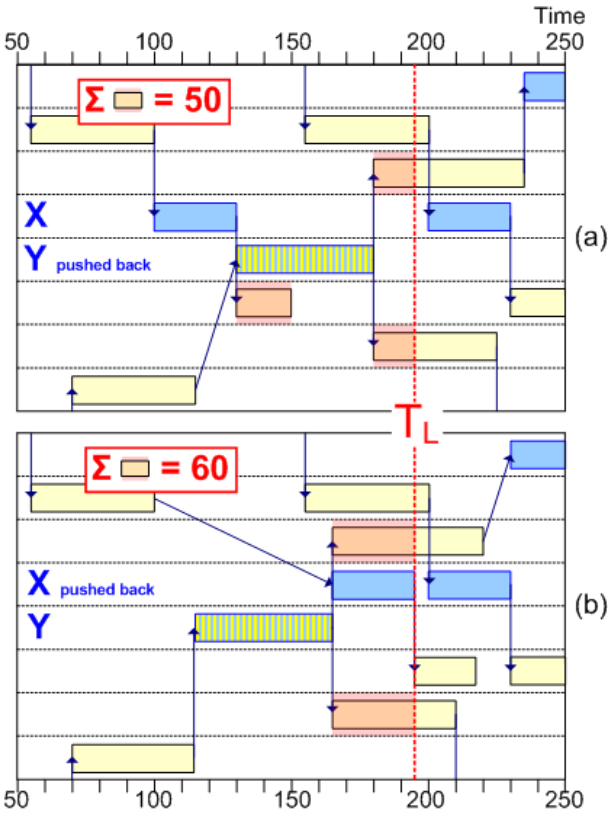


**Figure 4.** *Restricted subgraph search for two possible orderings and visualisation of the exploited parallelism.*

slot on a DSP. Lines 40ff are only executed when collisions occur on all DSPs: one *global* $T_{L,glob}$ is calculated from the collision on the first DSP in $L_i$ (line 40). Then the loop in line 50 applies the basic LEP method to all DSPs in the list with the time range $T_{L,glob}$. There is one mapping *and* ordering that provides the best exploitation of parallelism over all DSPs, which is chosen in line 90. Afterwards the schedule has to be updated accordingly (line 90). This extension obviously adds linear complexity in the order of $|L|$, that is the number of available processors.

The schedule updating, which follows the mapping in lines 10 and 90 can be performed in an efficient way by a thoroughly ordered breadth first search (BFS) downwards the subgraph below the process just moved: a general BFS pushes successors of a process into a queue, and iterates over this queue. When an queue element has been handled, it is popped from the queue. Following modifications allow for every vertex in the subgraph being only visited once and in the correct order: the first two candidates to start the BFS are apparently the two colliding processes *X, Y* in Fig. 4 corresponding to the `basic_LEP()` function (line 70). Every process that is a candidate for the BFS queue is checked whether it is not already in the queue and if so whether its end time (*X.end, Y.end* has changed. If its end time is unchanged, the subgraph of this process does not have to be considered for a rescheduling. i.e. reuse of the former schedule. If the end time has been changed, it is pushed into the queue, but not *necessarily* to its end. An ordering mechanism runs backwards through the queue and inserts the process such that the processes are ordered with increasing start times, taken from the former schedule. Thus it is ensured that all of a process' predecessors *and* those lying on the same DSP directly in front have already been rescheduled. The algorithmic complexity of the schedule update can be shown to be linear for sparse graphs.

## 7 Results

The algorithm has been tested on the same SDF graph sets as in our previous work [11], only enriched by process computation time lists for two DSPs instead of just one. The algorithms serving as benchmarks for the performance of this approach are the Highest Levels First (HLF) algorithm [8] and the the Earliest Task First (ETF) algorithm in [9]. To compare the behaviour and performance of the algorithms for one specific graph, we have to simulate the search space traversal of a partitioning algorithm by swapping processes back and forth between the HW and SW implementation.

Thus we obtain step by step new partitioning solutions $\pi$ and three schedule lengths $SL_\pi(\text{HLF})$, $SL_\pi(\text{ETF})$, $SL_\pi(\text{LEP})$. Furthermore, we calculate a lower bound $LB_\pi$ for the schedule length of each $\pi$. This lower bound is determined by the sum of the process computation times $pct_{ASIC}$, $pct_{ARM}$ or $pct_{SC}$, and the communication times along the critical path. For all considered partitioning solutions $\Pi_G$ of a graph $G$, the specific schedule lengths $SL_\pi$ of one partitioning solution $\pi \in \Pi_G$ are summed up, as well as the specific lower bounds, $LB_\pi$. Thus, we obtain for all three algorithms and the lower bound a global sum over all considered schedule lengths, e.g. for LEP:

$$SL_{global}(\text{LEP}, \Pi_G) = \sum_{\pi \in \Pi_G} SL_\pi(\text{LEP}). \qquad (1)$$

Fig. 5 shows a bar chart, in which the x-axis shows different graph sizes. The y-axis shows the global sums over schedule lengths for the three algorithms normalised to the global sum over the lower bound schedule lengths. The results have been averaged over 30 different graphs for each size ($|V|$ vertices in the SAG) and parallelism range $\varsigma$, if degree of parallelism $\gamma \in ]\varsigma - 0.5, \varsigma + 0.5]$, with $\varsigma \in \mathbb{N}, \gamma \in \mathbb{R}$. The degree of parallelism $\gamma$ is defined as the total number of vertices divided by the number of vertices in the critical path [15].

The algorithm outperforms both list scheduling techniques with higher margin considering larger graphs with

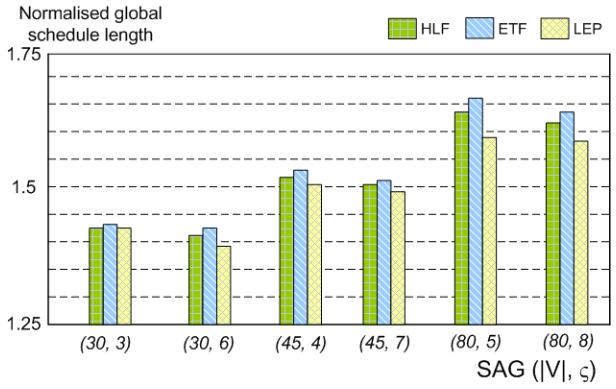


**Figure 5.** *Averaged global schedule lengths normalised to the global lower bound schedule lengths over different graph sizes.*

a higher $\gamma$. The runtime is four to five times higher than for the other algorithms. Although the relative improvement does not seem to be substantial, it has to be understood that the results are averaged over all partitioning solutions seen. In other words, for many partitioning solutions the algorithms return the same schedule, whereas for less partitioning solutions our approach returns a *much* better schedule. Partitioning algorithms relying on the list scheduling techniques would be repeatedly mislead by a much worse scheduling and most probably avoid the whole neighbourhood of the alleged bad partitioning.

## 8 Conclusion

In this work a multi-core extension of the fast rescheduling algorithm LEP [11], has been proposed. The main goal was the preservation of the linear algorithmic complexity for typical task graph sets in system design and a better performance than the list scheduling techniques usually used in HW/SW partitioning algorithms. Future work will focus on the subtleties of other objective functions, which consider varying power consumption of different DSPs and ensure a minimum processor load.

## References


[1] P. Arató, Z. A. Mann, and A. Orbán. Algorithmic Aspects of Hardware/Software Partitioning. 10(1):136–156, January 2005.

[2] K. S. Chatha and R. Vemuri. Magellan: multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs. In *Proc. of the 9th international symposium on Hardware/software codesign (CODES)*, pages 42–47, 2001.

[3] 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*, 2:5–32, 1997.

[4] M. Garey and D. Johnson. *Computers and intractability: a guide to NP-completeness*. W.H. Freeman, San Francisco, California, 1979.

[5] J. Grode, P. V. Knudsen, and J. Madsen. Hardware resource allocation for hardware/software partitioning in the lycos system. In *Proc. of Design, Automation & Test in Europe (DATE)*, pages 22–27, 1998.

[6] R. K. Gupta and G. D. Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design and Test of Computers*, 1993.

[7] J. Henkel and R. Ernst. An approach to automated hardware/software partitioning driven by high-level estimation techniques. Number 2, pages 273–289, 2001.

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

[9] 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.*, 18(2):244–257, 1989.

[10] A. Kalavade and E. A. Lee. The extended partitioning problem: hardware/software mapping and implementation-bin selection. In *Proc. of the 6th IEEE International Workshop on Rapid System Prototyping (RSP)*, page 12, 1995.

[11] B. Knerr, M. Holzer, and M. 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*, January 2006. in press.

[12] W. H. Kohler and K. Steiglitz. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *J. ACM*, 21(1):140–156, 1974.

[13] E. Lee and D. Messerschmitt. Static scheduling of synchronous data-flow programs for digital signal processing. In *IEEE Transactions on Computers*, volume 36, pages 24–35, 1987.

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

[15] Y. L. Moullec, N. B. Amor, J.-P. Diguet, M. Abid, and J.-L. Philippe. Multi-Granularity Metrics for the Era of Strongly Personalized SOCs. pages 674–679, March 2003.

[16] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA, 1989.

[17] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, 1993.

[18] W. Theerayod, P. Y. Cheung, and W. Luk. Comparing three heuristic search methods for functional partitioning in hardware-software codesign. 6(4):425–449, Sept 2002.

[19] F. Vahid and T. D. Le. Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning. pages 237–261, 1997.

[20] M. Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Trans. Parallel Distrib. Syst.*, 1(3):330–343, 1990.