Partial Task Assignment of Task Graphs under Heterogeneous Resource Constraints

Radoslaw Szymanek
Radoslaw.Szymanek@computer.org

Krzysztof Kuchcinski
Krzysztof.Kuchcinski@cs.lth.se

Department of Computer Science, Lund University
P.O. Box 118, SE 22100 Lund, Sweden

ABSTRACT

This paper presents a novel *partial assignment technique* (PAT) that decides which tasks should be assigned to the same resource without explicitly defining assignment of these tasks to a particular resource. Our method simplifies the assignment and scheduling steps while imposing a small or no penalty on the final solution quality. This technique is specially suited for problems which have different resources constraints. Our method does not cluster tasks into a new task, as typical clustering techniques do, but specifies which tasks need to be executed on the same processor. Our experiments have shown that PAT, which may produce non-linear groups of tasks, gives better results than linear clustering when multi-resource constraints are present. Linear clustering was proved to be optimal comparing to all other clusterings for problems with timing constraints only. In this paper, we show that, if used for multi-resource synthesis problem, as it is often used nowadays, linear clustering will produce inferior solutions.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms

Design

Keywords

constraint logic programming, task assignment, scheduling

1. INTRODUCTION

System synthesis is a design step which maps an initial specification into given architecture and decides its schedule. This can be done using task assignment and scheduling. During this step it is important to have an accurate model with a good abstraction level. It is often the case that an application is modeled using a coarse grain model which results in limited optimization possibilities. On the other hand, fine grain model can result in prohibitive large run-times where much time is spend on analyzing parts of the design search space which can give no or insignificant improvements to the final solution. Therefore, the designer should have possibilities to specify an application at a proper level and, only when run-times of the synthesis tools are of concern, the design space reduction methods should be used.

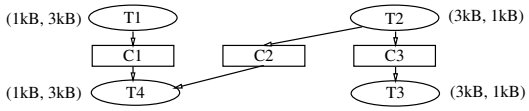

Figure 1: A task graph for the motivational example

In this paper, we present a novel method called *partial assignment technique* (PAT). PAT is able to improve efficiency of an assignment and scheduling as well as quality of the final results. This is possible since the complexity of assignment and scheduling problems require use of heuristics.

The complexity of task assignment and scheduling for heterogeneous architecture increases significantly when memory constraints are considered. These constraints define memory requirements for tasks and communications and therefore influence task assignment and scheduling decisions. The data memory aspect in embedded systems is especially important for signal and image processing applications which deal with enormous amounts of data. New synthesis methods are required to cope with these constraints efficiently. All techniques for reducing the search space without loosing (near)optimal solutions should be explored and our partial assignment technique is one of them.

The aim of this work is to develop efficient techniques for an embedded system synthesis tool that accepts system architecture description and an application specification given as a task graph and produces constraints which will reduce complexity of the assignment and scheduling. We assume that an application, specified in C-like language, is compiled into an acyclic task graph annotated with estimates of execution time, code and data memory requirements. PAT uses this task graph, generated from the original specification, as input. Therefore, the fine grain task graph can be used, which gives full optimization potential, but possibly long run-times of synthesis tools.

The rest of this paper is organized as follows. Section 2

DAC 2003, June 2–6, 2003, Anaheim, California, USA.

Figure 2: Target architecture

motivates our work through an example. In section 3, we outline related work in this area. Section 4 briefly presents our synthesis system MATAS system. Section 5 describes our partial assignment technique while section 6 presents experimental results. Finally, section 7 concludes the paper.

2. MOTIVATIONAL EXAMPLE

Consider a task graph depicted in Figure 1. This task graph consists of four tasks, T1, T2, T3, T4, depicted as ellipses and three data transfers between these tasks, C1, C2, C3, depicted as boxes. The code memory requirement for processor P1 (P2) for each task is represented as first (second) number in parentheses next to each task.

The data memory is used to store local data as well as input/output data (transmission buffers). Each task needs 4 KB of data memory during its execution. The data transfers between tasks send 2 KB of data. Each processor is equipped with 3KB of code memory and 8KB of data memory.

The goal of system synthesis is to make assignment of tasks to processors and schedule all tasks and communications. The resulting assignment and schedule should have minimal deadline and fulfill code and data memory constraints. The problem can be solved using constraints programming over finite domain [6]. In this constraint programming approach we first specify decision variables and constraints over these variables and then solve constraint problem by assigning values to the decision variables. In the process of variable assignment the constraints propagate changes and reduce the search space. Different assignment methods can be used. In this paper, we use our own heuristic since branch-and-bound methods can be used for relatively small size problems only.

In our example, each task Ti has two decision variables, namely S_{Ti} and P_{Ti}, which denote task start time and processor assigned for a given task. All these decision variables are represented by finite domain variables (FDV). In our problem, decision variables S_{Ti} have a finite domain and can take a value from range 0 ms to 10 ms. Processor assignment variables, P_{Ti}, have value 1 or 2, which denote a processor number for execution of a given task. There are also decision variables for communications in addition to task related decision variables. Each communication Ci has its start time and duration denoted by S_{Ci} and D_{Ci}.

When all decision variables and their domains are specified, we need to state all constraints which must be satisfied in the final solution. In our example, we have typical precedence constraints between tasks and communications as well as resource constraints for processors and buses. We also have code memory constraints which ensure that there is enough memory on each processor for assigned tasks. Finally, data memory constraints ensure that enough data memory is available for communication buffers and for local data of tasks [8]. For example, precedence constraints are defined as inequalities and for task T1 and communication C1 they are specified as $S_{T1} + D_{T1} \leq S_{C1}$, meaning that the time of start of communication C1 must be at least equal the time when task T1 finishes its execution.

The specification of these constraints will already restrict some values of FDV's. However, most of FDV's will still have more than one possible assignment. During search procedure different values of FDV's are evaluated until a valid assignment of values to FDV's is found. Our technique reduces the number of possible values of FDV's. Therefore it speeds up, the most time consuming step, search procedure.

Consider a target architecture as depicted in Figure 2. It consists of two processors connected by a bus. There are few valid task assignments and schedules for a task graph depicted in Figure 1. An optimal schedule of a given task graph is presented in Figure 3. This schedule requires only C2 communication to be scheduled on the bus and therefore reduces bus utilization and memory requirements. Tasks T1 and T4 as well as T2 and T3 communicate using their local memories and additional memory buffers for communication need to be assigned only for communication C2, between tasks T2 and T4. Additional constraints, identified by PAT, $P_{T1} = P_{T4} \wedge P_{T2} = P_{T3}$ are added to enforce that these tasks are executed on the same processor, and durations of communications are zero $D_{C_{T1,T4}} = 0 \wedge D_{C_{T2,T3}} = 0$. These constraints facilitate achievement of good quality schedules under all resource constraints. They do not define the final assignment but state that selected tasks need to be executed on the same processor.

Consider a chain of assignments of a value to a FDV which needs to be made to arrive to the final solution as presented in Figure 4. Each of these assignments can be illustrated as a decision node (e.g., in branch-and-bound algorithm) annotated with the number of possible decision branches. Our PAT reduces the search space by reducing the number of these decisions. This is depicted, for our example, by smaller numbers in parentheses next to P_{T3} and P_{T4} nodes. These two nodes represent decisions on which processor tasks T3 and T4 are executed. In addition, since two communications are done locally the number of possible decisions at S_{C1} and S_{C3} nodes is also reduced.

For this particular example, an efficient linear clustering is not possible. A clustering of tasks T1 and T4 as well as T2 and T3 produces two new clustered tasks. All valid assignments and schedules of these clusters result in executing cluster containing tasks T2 and T3 first and then cluster of tasks T1 and T4, thus producing an inefficient solution. Other clusterings will violate code memory constraints.

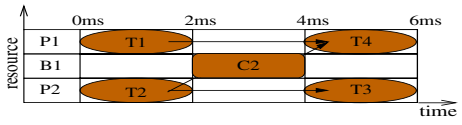

Figure 3: Possible schedule

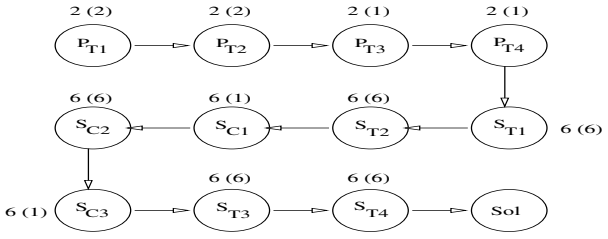

Figure 4: Search Assignment Ordering

3. RELATED WORK

The related work which tries to reduce the size of the problems for different synthesis activities, such as assignment and scheduling, usually concentrates on clustering. This technique tries to select tasks which are close to each other and builds a new task which contains all selected tasks. Our method is different since it does not build new tasks but instead adds new assignment constraints.

The work presented in [5,7,10] concentrates on parallelization of software systems in multiprocessor homogeneous environment through usage of clustering techniques. These techniques reduce also the complexity of the assignment problem but they actually change the input problem since they create new tasks to represent clusters. They have also assumed a number of restrictions. For example, they cannot cluster an application into a given number of clusters or they ignore network contention.

Mapping of heterogeneous task graphs into heterogeneous architectures was addressed in [3]. The core assumption of this work is that a speed of all processors is equal to a baseline processor speed multiplied by a constant. Both the application task graph and the architecture task graph are clustered separately. This clustering produces two multi-layer clustered graphs, which levels are later matched against each other. The clustering of the task graph examines first fork nodes. Afterwards remaining tasks are considered and added into the existing layered graph. They claim that mapping complexity is reduced without loss of quality due to a-priori multi-layer clustering of task and architecture graphs.

The critical path-based clustering procedure was presented in [1]. It takes into account different execution time of tasks depending on the processor. However, it does not consider code and data memory during clustering. It creates clusters of tasks which need to be later executed as single tasks.

Our approach makes it possible to obtain complexity reduction of assignment and scheduling problem for the heterogeneous architecture in the presence of multi-resource constraints. All of the mentioned approaches addressed a simple case when the only resource is time. Our method does not simplify the architecture by introducing one homogeneous communication structure or homogeneous computing environment. We also do not assume that there is a speed factor to which the speeds of all processors has to be referred to. Our approach does not produce clusters, it produces constraints which reduce the possible assignment of tasks to processors. In addition all clustering approaches which do not create linear clusters run into a danger of creating designs which will eventually deadlock [5]. Our approach avoids deadlocks since it does not cluster tasks but constraints task assignment within a group of tasks.

Traditional clustering combines tasks into a new task which is later treated by synthesis method as an ordinary task. The theoretical work on clustering proves the superiority of linear clusters [4]. This does does not apply here since we have not only timing constraints but also code and data memory constraints. The task assignment and scheduling for such systems are closely coupled problems that they should be solved together. This is enabled by our methodology.

4. MATAS

The proposed framework has been written entirely in Java and it uses our own JaCoP (Java Constraint Programming)

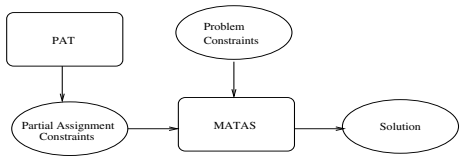

Figure 5: MATAS framework and PAT

engine to model and solve synthesis problems. The important advantage of our approach is the gradual refinement of the model through addition of new constraints. Since the system can handle heterogeneous constraints, the refinements can be very specific. In particular, constraints can specify on which processor a task should be executed, based on the assignment of other tasks. They can also specify which communications need to be local. The addition of new constraints (refinements) decreases the search space by making selected decisions explicit to the solver. This decreases the number of search nodes or removes some of the branches in case of branch-and-bound search algorithm. Note, that in our case we do not use branch-and-bound algorithm but decisions made by our heuristic are simplified.

In our approach, JaCoP (a constraint programming solver) is used to model the system architecture, the application and the synthesis problem. A general introduction to CLP is given in [6]. Briefly, a CLP program consists of constraints over finite domain variables and a search method. Each finite domain variable (FDV) is initially defined by a set of integer values that constitute its domain. Constraints specify relations among these variables. A constraint engine provides constraint consistency and propagation methods. Therefore, restricting a domain of one FDV propagates to other FDV's and usually results in restricting domains of the other FDV's. Partial task assignment constraints improve propagation as well as cut off some parts of the search space.

The MATAS synthesis system [8] makes both task and communication assignment and scheduling. It considers timing constraints as well as code and data memory constraints. The goal of the system is to find (near) optimal solution, in respect to schedule length while fulfilling all constraints. The synthesis is done within constraint programming framework, as sketched in section 2 and presented in Algorithm 1. This algorithm tries to use different resources, such as time, code and data memory, evenly. The decisions are made based on estimates of future use of these resources. Both time and data metrics, which are used to choose next task to schedule, reflect the usage of those resources in relative terms. Often the next task will increase either critical path length or data memory usage, and therefore it is important to know which resource is currently more used and act accordingly. Since the algorithm is constraint-driven the result of all decisions directly propagate to all FDV's and constraints. This makes the implementation of different search heuristics easier and less error-prone.

The presented partial assignment technique is an extension of the prototype design system MATAS, as presented in Figure 5. Our method introduces new constraints which limit the possible assignment of the tasks. The idea of these constraints is to guide MATAS system towards better solutions. PAT constraints (1) state that all tasks within the same group are assigned to the same processor and the related communication between these tasks has duration zero.

$$\forall T_i \in G_n \forall T_j \in G_n : P_{Ti} = P_{Tj}, D_{C_{i,j}} = 0 \quad (1)$$

We do not create new tasks from old ones but specify which tasks need to be assigned to the same processor. Up to authors best knowledge this is the only solution of this type which reduce the complexity of the task assignment and scheduling problem without changing application model.

Algorithm 1 The general idea of MATAS algorithm.

$\mathcal{R} \leftarrow$ *Tasks without predecessors*
while $\mathcal{R} \neq \emptyset$ **do**
  select T_{time} with minimal mobility
  select T_{data} with greatest
  $\sum_{O \in S(T_{data})} O.size() - \sum_{I \in P(T_{data})} I.size()$
  {$S(Task)$ denotes all data produced by the task}
  {$P(Task)$ denotes all data consumed by the task}
  {size() denotes the size of the given data}
  if $metrics_{data} > metrics_{time}$ **then**
    $nextTask = T_{data}$
  else
    $nextTask = T_{time}$
  {Task which decreases usage of more utilized resource has been chosen}
  for all processors which can execute $nextTask$ **do**
    find minimum $S_{nextTask}$
    compute time usage factor t_u for $nextTask$
    compute code usage factor c_u for $nextTask$
    compute data usage factor d_u for $nextTask$
  choose processor P_{min} which minimize $t_u + c_u + d_u$
  schedule $nextTask$ on P_{min}
  schedule all incoming communications and reserve communication buffers
  $\mathcal{R} \leftarrow \mathcal{R} \setminus \{nextTask\} \cup$
    {*tasks with all input data available*}

5. PARTIAL ASSIGNMENT TECHNIQUE

The system synthesis has to take into account multiple resources. In our model, we have currently three types of resources for which parallel tasks compete: time slots, code and data memories. We assign tasks to processors time slots and communication tasks to bus time slots. Each task needs data memory during execution as well as produces and consumes data which are also stored in data memory. The code memory needs to be reserved for each task so it can be executed on a selected processor. The complete solution specifies these three assignments. Since the number of decision variables is normally large, the size of the search space can be huge. Our method makes selected assignment decisions explicit by specifying assignment constraints (1).

PAT makes partial assignment decisions based on several closeness measures which reflect resources use, such as time, data memory and code memory, between groups of tasks. The final closeness measure is defined as a weighted sum of these closeness measures as defined below. Smaller numbers represent closer groups.

$$closeness_{gi,gj} = w_1 \cdot ct_{gi,gj} + w_2 \cdot cc_{gi,gj} + w_3 \cdot cd_{gi,gj} \quad (2)$$

where w_1, w_2 and w_3 are weights, and $ct_{gi,gj}$ is the closeness measure for time, $cc_{gi,gj}$ is a closeness measure for code memory, and finally $cd_{gi,gj}$ is a closeness measure for data memory. In our experiments all weights are equal.

There are two crucial assumptions when computing closeness measures. The values in the dominator in (3), (4), and (5) are always computed under an assumption that groups g_i and g_j execute on different processors. On the other hand, the numerator is the minimal value under assumption that both groups execute on the same processor. Each of the metrics may have a value lower than one and this will indicate that there is a possible gain if both tasks groups are executed on the same processor.

The closeness measure for time resource is given below

$$ct_{gi,gj} = \frac{\min_{P_{gi}=P_{gj}} D_{gi} + D_{gj}}{\min_{P_{gi}\neq P_{gj}} D_{gi} + D_{gj} + C_{gi,gj}} \quad (3)$$

where D_x denotes the execution time of group x, and $C_{x,y}$ denotes the communication time between group x and y. The closeness value will be smaller with more communication time required to communicate data between two groups. Small closeness value $ct_{gi,gj}$ indicates that there will be a gain in schedule length if both groups are executed on the same processor.

The second closeness measure is for code memory

$$cc_{gi,gj} = \frac{\min_{P_{gi}=P_{gj}} \frac{CM_{gi}+CM_{gj}}{CM(P_{gi})}}{\min_{P_{gi}\neq P_{gj}} \frac{CM_{gi}}{CM(P_{gi})} + \frac{CM_{gj}}{CM(P_{gj})}} \quad (4)$$

where CM_x denotes how much code memory is required by a group x, and $CM(P_x)$ denotes the amount of code memory available at the processor which executes group x. The minimal relative usage of code memory under the assumption that groups execute on the same processor is represented by numerator. This will be divided by the sum of minimum relative usage of code memory for each of the groups executing on different processors. The closeness function for code memory reflects, in relative terms, how much more code memory would be used if two groups were grouped. Since the code memory is a resource which is reserved for the whole time it is possible to use this type of metrics.

The most difficult resource to take into account is the data memory. Equation (5) presented below defines this measure,

$$cd_{gi,gj} = \frac{\min_{P_{gi}=P_{gj}} \frac{DM_{gi}+DM_{gj}}{DM(P_{gi})}}{\min_{P_{gi}\neq P_{gj}} \frac{DM_{gi}}{DM(P_{gi})} + \frac{DM_{gj}}{DM(P_{gj})} + \frac{DM(C_{gi,gj})*C_{gi,gj}}{min(DM(P_{gi}),DM(P_{gj}))}} \quad (5)$$

where DM_x denotes how much data memory is needed for group x temporary data. This is specified by (6).

$$DM_{Gx} = \sum_{T_y \in G_x} DM_{Ty} * D_{Ty} \quad (6)$$

The tasks use data memory temporarily so we need a different approach to compute closeness function for this resource type. Each task is annotated with local data memory size multiplied by its execution time. For a given group all data memory requirements are added and divided by the data memory size of the processor. In case when two tasks are executed on different processors, an additional cost appears due to double reservation of data memory buffers for communication. This cost is represented by a third term in denominator of equation (5). Since the communication time can differ we assume that communication time is the average of possible communication times. This cost is normalized by

the smallest data memory size of one of the processors executing both groups.

The PAT decisions are difficult since the knowledge on assignment of tasks to processors is not available yet. It is possible that grouping of two tasks will increase the schedule length. Therefore, each resource closeness measure aims at reflecting the possible degradation or improvement of resource usage. Our PAT algorithm, presented in Algorithm 2, initially starts with one task per group. It then computes closeness measures for each pair of groups with a non-local communication. Each PAT iteration will merge two closest groups thus making at least one communication local. This communication will not require bus access. The algorithm will stop when expected reduction of task graph is reached.

Algorithm 2 PAT algorithm.

for all i **do**
  $G_i = \{T_i\}$
$tasks\_to\_constraint = \frac{|\{T_i\}|}{reduction factor}$
while $tasks\_to\_constraint > 0$ **do**
  for all non-local communications C_i **do**
    G_p = producer(C_i)
    G_c = consumer(C_i)
    $clss_i = closeness_{G_p,G_c}$
  select C_i with smallest $clss_i$
  merge groups G_p and G_c
  make all communication between merged groups local
  $tasks\_to\_constraint = tasks\_to\_constraint - 1$

6. EXPERIMENTAL RESULTS

Our technique was applied to a real-life example presented in [9]. Their problem is, however, quite simplified from our perspective since the application is mapped onto a homogeneous multiprocessor architecture consisting of 7 processors and a single bus. The authors also do not take into account code memory and precedence constraints. However the application consists of large number of tasks and communications which makes it a good benchmark example. Our MATAS/PAT system has been applied to this example and produced the results presented in Table 1.

The first experiment has no partial assignment constraints introduced by PAT. Therefore there is no problem complexity reduction and it has a full optimization potential. Since the full search has not been performed the obtained solution is not proved optimal. The lower bound for deadline of this example, with precedence constraints and a given architecture, is equal to 1975 ms. The first and the second experiment produce solutions that are ∼1% worse than a lower bound and might be optimal.

In the following experiments we compare linear clustering with PAT. Linear clustering uses the same metrics as PAT for making clustering decisions. However, the clustering decisions for some tasks, despite metrics indication, has been rejected when they produced non-linear clusters since they might create deadlocks.

Table 1: Experimental results for real-life example.

Exp	Method	Tasks constrained	% of all tasks	Deadline [ms]	Bus Load [ms]
1.	MATAS	0	0	1992	176
2a.	PAT	30	25	1992	176
2b.	Clustering	30	25	1992	176
3a.	PAT	60	50	2163	56
3b.	Clustering	60	50	2013	132
4a.	PAT	90	75	2395	2
4b.	Clustering	90	75	2394	130

In the second experiment 25% of tasks have been partially assigned by PAT or clustered. In this particular case, a solution obtained with both PAT constraints and linear clustering give the same solution as in the previous case. The problem complexity reduction has been achieved without penalty on a quality of the solution.

In experiment three and four both PAT and clustering simplified the problem at the expense of the achieved deadline. However the obtained deadline still lies within 25% from the lower bound when the complexity of the assignment problem was reduced by ratio 50% or 75%.

In experiment three the clustering with MATAS obtained shorter deadline than PAT with MATAS. In this case, however, it was also checked that the constraints imposed by PAT do not exclude the solution found by MATAS with clustering. In this particular case, clustering guided the MATAS system better. This real life example shows that for problems where only time constraints are imposed the linear clustering will always give as good result as non-linear methods like PAT. It conforms to the theory presented in [4]. This however is not the case for multi-resource problem as indicated in further experimental results.

Our technique has also been evaluated on random task graph examples. In this case, we can fully evaluate our method with specific code and data memory requirements as well as use heterogeneous architecture which is depicted in Figure 6. The architecture consists of four heterogeneous processors and three buses. The speed of bus B1 is 2MB/s and the speed of other buses is 1MB/s. The experiments were performed on task graphs generated by TGFF [2]. The options supplied to TGFF enforced the average execution time, code memory requirement, temporary task data, and application data to be equal respectively to 4 ms, 4kB, 3MB, and 5MB. They have also respectively constrained the deviation of those parameters to 2ms, 2kB, 1MB, and 2MB. The task graph has maximum number of incoming and outcoming edges equal to 2. Each graph consists of 80 tasks and a number of communications between 110 and 130. They can be regenerated using TGFF and numbers 1 to 20 as random seeds. The task graph characteristics make them difficult to map and schedule on the test architecture. The experiments results, represented as a line in Table 2, represent the average values obtained by solving 20 different task graphs.

The first experimental setup (1) uses the target architecture where each processor has 100KB of code memory and 100MB of data memory. The maximum execution latency for the task graph has been set to be at most 350 ms, which was then gradually reduced to find best shortest deadline. The MATAS system was used to solve these problems in three different settings: without any additional constraints

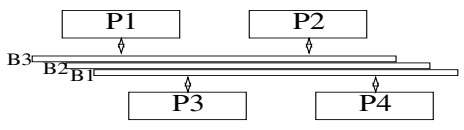

Figure 6: Experimental Architecture

Table 2: Experimental results - random graphs

Exp.	Dead-line [ms]	Code Mem. Util. [%]	Code Mem. Peak [%]	Data Mem Util [%]	Data Mem. Peak [%]	CPU Util. [%]	Bus Util. [%]
1. M	144	73	91	39	95	43	46
1. C	137	71	90	43	96	48	41
1. P	131	73	91	40	96	52	42
2. M	158	88	99	52	97	42	45
2. C	151	91	99	58	96	45	37
2. P	144	93	100	54	95	48	40

(M), with clustering (C) and finally with PAT constraints (P). The experiments with PAT and clustering used best design space reduction ratio from the real case experiment of 25% and assignment of one fourth of 80 tasks has been constrained. In this particular experimental setup, MATAS with PAT constraints achieved ∼4% reduction in average deadline comparing to best results achieved by MATAS with clustering.

The reduction was partially obtained through executing tasks on processors which required more code memory, thus larger code memory utilization has been noticed. The sum of data memory allocated at different processors in the system, represented by data memory utilization has been reduced. This is due to better placement of data in data memory across the whole architecture. In general PAT constraints helped to find more parallel solutions both in terms of computation as well as communications.

In the next experiments (2) we have reduced the amount of available memory on all processors. Each processor has 75MB data memory and 75KB of code memory. The same deadline as previously found for given solutions was applied and it was gradually extended for cases when the deadline could not be fulfilled. It can be noticed that the processor utilization ratio as well as bus utilization ratio has dropped since the application cannot always be executed with the same degree of parallelism as previously.

The memory resources utilization increased when compared to the previous experiments. The data memory utilization is higher since less resources are available. There are two important factors which make data memory a bottleneck, even if the value of data memory utilization around 50% may not suggest this. First, when the code memory of a given processor has been already used, we are not able to assign more tasks to this processor. Second, data is not stored evenly over the whole execution time. Often almost the whole data memory at a single processor is used temporarily for communication buffers which is reflected by very high data memory peak. This phenomenon will restrict execution of new tasks on this processor since no more data memory is available at this moment. The code memory influence was reflected by high averaged peak of 100% and an average code memory utilization of 90%. The average deadline has increased because tasks from the critical path cannot be executed until other non-critical task consumes data and makes more data memory available. It increases also in the case when the fastest task implementation cannot be executed due to code memory restrictions.

Both experiment setups show that reduction of the search space resulted, in some cases, in decreased quality of the solution. The deadline was increased slightly. However, consistent reduction of the heuristics runtimes equal to the percentage of constrained tasks was achieved. The PAT method does not transform the problem itself, it just adds partial assignment constraints. Clustering on the other hand transforms the problem and therefore it influences not only task assignment but also scheduling of communications. The clustering itself did help to improve the solution quality comparing to original MATAS approach but these solutions are usually not as good as PAT can deliver.

7. CONCLUSIONS

Our partial assignment technique, presented in this paper, makes it possible to decrease the complexity of the assignment and scheduling problem. During the search, assignments of all tasks, within the same group, to the processor is performed only once. This method works for heterogeneous architecture with heterogeneous communication structure. The architecture resources can be of different nature, from simple ones, such as code memory, to more complex, such as computation time or data memory. The experimental results indicate that PAT can simplify the problem by removing inferior parts of the search space, which was observed in the second experiment setup of real-life example.

Our heuristic is able to improve quality of the scheduling and assignment as it was observed in both random experimental setups. It gives better results comparing to clustering as well as not pre-constrained original MATAS approach.

Acknowledgment

This work was partially supported by Swedish Foundation for Strategic Research within INTELECT project. It was also supported by Marie Curie Host Fellowship within HPMT-2000-00031 project at IMEC.

8. REFERENCES

[1] G. L. Bharat P. Dave and N. K. Jha. COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7:92–104, March 1999.

[2] R. Dick, D. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Sixth International Workshop on Hardware/Software Codesign)*, pages 97–101, 1998.

[3] M. M. Eshagian and Y. C. Wu. Mapping heterogeneous task graphs onto heterogeneous system graphs. In *Heterogeneous Computing Workshop*, pages 147–160, 1997.

[4] A. Gerasoulis and T. Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems*, 4:686–701, June 1993.

[5] D. Kadamuddi and J. J. Tsai. Clustering algorithm for parallelizing software systems in multiprocessors environment. *IEEE Transactions on Software Engineering*, 26:340–361, April 2000.

[6] K. Mariott and P. Stuckey. *Introduction to Constraint Logic Programming*. The MIT Press, 1998.

[7] M. Senar, A. Ripoll, A. Cortes, and E. Luque. Clustering and reassignment-based mapping strategy for message-passing architectures. In *Parallel Processing Symposium and Symposium on Parallel and Distributed Processing IPPS/SPDP*, 1998.

[8] R. Szymanek and K. Kuchcinski. A constructive algorithm for memory-aware task assignment and scheduling. In *Ninth International Symposium on Hardware/Software Codesign*, April 2001.

[9] C. M. Woodside and G. G. Morton. Fast allocation of processes in distributed and parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 4:164–174, February 1993.

[10] M.-Y. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1:330–343, July 1990.