Run-Time Mapping of Applications to a Heterogeneous Reconfigurable Tiled System on Chip Architecture

Authors, affiliation, acknowledgement and some references removed
for the purpose of anonymous reviewing

***Abstract*— This paper describes the implementation and evaluation of an algorithm that maps a number of communicating processes to a heterogeneous tiled System on Chip (SoC) architecture at run-time. The mapping algorithm minimizes the total amount of energy consumption, while still providing an adequate Quality of Service (QoS). The properties of the algorithm are described and evaluated and a realistic mapping example is given.**

***Index Terms*— mapping, system on chip (SoC), run-time, tiled, heterogeneous, reconfigurable, architecture, drm**

I. Introduction

The architecture of a portable multimedia system has to meet many conflicting requirements. For example, it should be energy-efficient, due to the scarce energy resources and it should be flexible. It should be flexible so that it a) can employ a lot of different standards, b) can be adapted quickly to implement a new standard c) can run different sets of tasks concurrently and d) can adapt to the dynamically changing environment.

The designer can choose from a wide spectrum of architectures to implement such a system. This can vary from energy-efficient, high-performance but static and inflexible ASICs to flexible and easy programmable but energy hungry general purpose processors. The optimal choice depends on the application/algorithms and several other aspects, including the available energy budget, the time to market and the production volume.

However, no specific architecture will meet all these requirements perfectly. A heterogeneous System on Chip (SoC) with different kind of (reconfigurable) processing tiles interconnected by a Network on Chip (NoC) as depicted in the lower part of Figure 1 provides a nice solution for this dilemma. Examples of different types of processing tiles are:

- General Purpose Processor (GPP), e.g. ARM,
- Digital Signal Processor (DSP)
- Application Specific Integrated Circuit (ASIC),
- Domain Specific Reconfigurable Hardware (DSRH) e.g. Montium [**?**]
- Field Programmable Gate Array (FPGA), e.g. embedded FPGA's

The best of both worlds (energy-efficient and flexible) can be combined in such a heterogeneous architecture. For example, small computational intensive algorithms of an application can be mapped to an ASIC or a coarse reconfigurable tile avoiding a power hungry tile such as a general purpose processor. On the other hand, control intensive but computational not so intensive parts of the application can be mapped better to a general purpose processor. In this way, the architecture can match the application instead of the other way around, as usual.

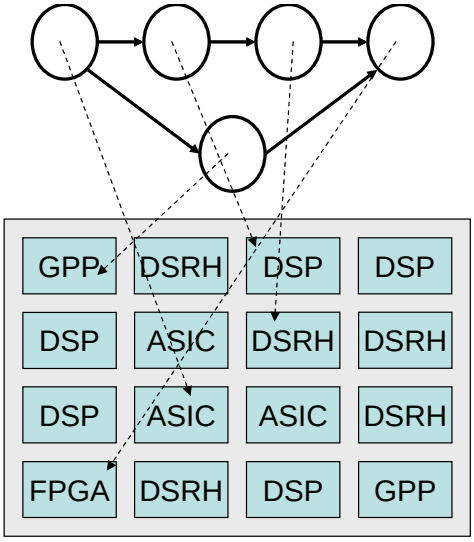

Fig. 1. SoC Template and the Mapping of a Process Graph

Such a heterogeneous tiled architecture has also many other advantages. To name a few: a) tiles of the same type can be duplicated when the number of transistors grow in the next technology round, b) replication of tiles eases the verification process, c) tiles do not grow in complexity with a new technology, d) relative small tiles makes it possible to optimize them extensively, e) computational performance scales about linearly with the number of tiles, f) unused tiles can be switched off to reduce the energy consumption of the chip, g) locality of reference is exploited, h) it is possible to have individual clock domains per tile and for reconfigurable tiles it is possible to do partial dynamic reconfiguration on a per tile basis.

However, the use of such a heterogeneous tiled SoC architecture changes the standard development flow (e.g. code a program in C and compile or code functionality in VHDL and synthesize). The designer has to partition the application into a graph with communicating functional processes (see top of Figure 1). In a process graph, a vertex represents a functional process and an directed edge represents communication between functional processes. For each functional process one or more

realization(s) for one or more different types of processing tiles have to be made. Designing more, functional equivalent, realizations of the same process for different types of tiles makes it possible to run an application even when the most optimal tiles are not available. Often, the partitioning of an application into a process graph arises naturally from the application; an example of a functional process is an FFT. Quite often, the designer knows which kind of realizations might make sense. For example, bit level processes such as turbo/convolutional encoding or random number generation match very well to a FPGA or an ASIC. Computational intensive word based algorithms such as an FFT can be mapped to an ASIC, DSP or coarse reconfigurable architecture while algorithms with a lot of control construct, such as `if()` .. `then` .. `else` .., `while` () loops, can be mapped best to a general purpose processor. The designer plays an important role in this process and we assume that the partitioning and the choice of possible realizations are still made manually by the designer.

The mapping of these realizations to the heterogeneous tiled SoC architecture can best be done automatically at run-time. At design time it is not known which applications run simultaneously and how the external environment (with regard to available services, end-user behavior, wireless link quality) behaves. Therefore, this mapping decision has to be made at run-time. This article describes an implementation and evaluation of such a run-time mapping algorithm.

Section II describes related work. Section III gives an introduction of the MinWeight algorithm that finds an optimal mapping solution of the available process realizations to the tiled SoC architecture. Section IV describes the implementation of this algorithm and analyzes the (strong and weak) properties of the MinWeight algorithm. Furthermore, some adaptations are proposed to get a mapping which fulfills some additional constraints. Section V presents a model of the mapping problem again to give an optimal solution under additional constraints. Section VI gives an example of the mapping of the digital baseband part of a radio receiver. First we describe shortly the application followed by the results of the MinWeight algorithm. Section VII concludes the article.

II. Related Work

In the area of scheduling and optimization theory (operations research) a lot of literature exists on models which have some similarities with the considered problem (see e.g. [6], [2]). However, our application has some properties, which not allow us to use the existing approaches without modification. Compared to traditional scheduling for parallel systems we have the following differences:

- use of a heterogeneous architecture instead of a homogeneous architecture.
- the most important optimization parameter is minimization of the energy consumption instead of performance. The goal of most scheduling methods is to optimize the performance. In our method, the required performance is only one of the constraints, which has to be satisfied.
- the communication is an important parameter to be included in the total optimization because the communication consumes a substantial part of the total energy budget. In normal multiprocessor systems, the main focus is on the computation costs.

Another important difference with regard to optimization in literature is that we need to have a light-weight algorithm. It may be better to have to reasonable good solution computed with little effort than to have an optimal solution that requires a lot of effort for its computation. Therefore, a lot of existing optimization algorithms are on beforehand not acceptable for us.

Hu and Marculescu [5] address energy-aware communication and task scheduling for NoC architectures under real-time constraints. Their scheduling method is based on the statistical analysis of the different processes. However, for simple process graphs with vertices of low degree (as is often the case for typical process graphs) we are able to compute an optimal solution with the MinWeight algorithm, while the Hu approach gives a sub optimal solution. One of the problems that we foresee with the Hu approach is that it is difficult to add additional constraints, such as a limited capacity of processors. Hu assumes an infinite capacity of the processors, which is not realistic in practice.

III. MinWeight Algorithm

This section describes the MinWeight algorithm that determines the weight of a minimum processor assignment for *any* weighted process graph G_w and a set of processors P. Its running time is exponential. However, in practice it can compute solutions quite fast, as long as the input graphs have only a small number of vertices with a high degree (greater than two). A proof of the correctness of the algorithm, the complexity of the algorithm and further explanation can be found in [**?**].

A. Preliminaries

For the modeling of the optimization problem mentioned in the Introduction we consider simple graphs, denoted by $G = (V_G, E_G)$, where V_G is a finite nonempty set of vertices and E_G is a set of unordered pairs of vertices, called edges.

For a vertex $u \in V_G$ we denote its neighborhood, i.e. the set of adjacent vertices, by $N(u) = \{v \,|\, (u, v) \in E_G\}$. The *degree* $\deg(u)$ of a vertex u is the number of edges incident with it.

If $e = (u, v) \in E_G$, the *contraction* G/e of G is the simple graph obtained from G by replacing u and v and the edges incident with u and v by one new vertex uv and edges joining uv with the vertices adjacent to u or v in G.

Now let $G = (V_G, E_G)$ be a simple graph with vertex set V_G and edge set E_G. The vertices of G represent the tasks that have to be performed on a set P of processors. An edge $e = (u, v)$ exists if and only if there is communication between the processes of task u and task v.

Let the vertex weight $w_p^u \geq 0$ represent the costs of processing task u on processor $p \in P$. This way a weight vector w^u of size $|P|$ is defined for $u \in V_G$. If in practice u cannot be performed on processor p, this can be expressed by setting $w_p^u = \infty$ (or a bounded, sufficiently large number M).

For $e = (u, v) \in E_G$ let the edge weight $w_{pq}^e \geq 0$ represent the communication costs between the processes of task u and v, if u is performed on processor $p \in P$ and v is performed on processor $q \in P$. This way a matrix W^e of size $|P| \times |P|$ is

defined for $e \in E_G$. If in practice two tasks u and v cannot be performed on the same processor p simultaneously, we can model this by adding an edge $e = (u, v)$ and setting $w_{pp}^{(u,v)} = M$, where M is a sufficiently large number.

The graph G together with the weight vectors w^u, and weight matrices W^e is called a *weighted process graph*, and denoted by G_w.

We call a mapping $f : V_G \rightarrow P$ a *processor assignment* of G. Let $\mathcal{F}_G$ denote the set of all processor assignments of G. We define the *weight of a processor assignment* $f \in \mathcal{F}_G$ for a weighted process graph G_w as

$$w(f) = \sum_{v \in V_G} w_{f(v)}^v + \sum_{(u,v) \in E_G} w_{f(u)f(v)}^{(u,v)}.$$

A *minimum weight processor assignment* f^* is a processor assignment that has minimum weight, i.e., with $w(f^*) = \min\{w(f) \mid f \in \mathcal{F}_G\}$.

The MINIMUM WEIGHT PROCESSOR ASSIGNMENT problem (MWPA) is the problem of finding a minimum weight processor assignment for a given weighted process graph G_w and a set P of processors.

If any minimum weight processor assignment f will map task u on processor q, we say that u is *fixed* on q. If $w_p^u = 0$ for all processors $p \in P$, we will say that u is *free*.

B. MinWeight Algorithm

Figure 2 shows the MinWeight algorithm. In each iteration of the loop over the steps **(2)** to **(6)**, one vertex is removed from the graph and the weight of the vertex and the accompanying edge(s) are added to another vertex of edge of the graph. Depending on the degree of the vertex, a different action is performed. Step **(3)**, **(4)** and **(5)** take care of a vertex with degree 1, 2 and 3 or higher respectively. As an example, we give a short explanation of the action performed in step **(3)** for a vertex v of degree 1. First, the neighbor vertex u of vertex v is identified. There is at most one neighbor, because vertex v has degree 1. Next, for we pick a processor p for vertex u and perform the following evaluation: For which processor q for vertex v do we get a minimum for the cost for the processing of v on q and the costs for the communication between p and q. This minimum weight is added to the weight of processing u on processor p. This is done for all possible processors for vertex u. After evaluation of all these possibilities, vertex v is removed from the graph.

IV. EVALUATION

This section describes the implementation of the MinWeight algorithm of Section III, the properties of the algorithm and a description of a slightly adapted version of the algorithm to cope with one of the limitations of the MinWeight algorithm.

A. Implementation

The MinWeight algorithm is implemented in C++ using the Boost Graph Library [1] (BGL). The BGL is a library that is specially developed for the representation and manipulation of graphs. In the current implementation, it is possible to read a process graph from a file, to run the MinWeight algorithm and to output the results in a nicely formatted file.

In the implementation, the weight W for an edge is composed from two different elements because the weight depends on two quite different properties. First, the costs of communication between two different processors on the SoC. This is an architectural property of the SoC, which should be supplied by a SoC model. Second, the amount of communication, which is a functional property of the application that is specified in the process graph.

B. Properties of the MinWeight Algorithm

In this part we describe and discuss the most relevant strong and weak points of the MinWeight algorithm with respect to our specific mapping problem.

Firstly, the MinWeight algorithm computes the optimal solution to the mapping problem instead of an approximation. This is a strong advantage of the algorithm.

Secondly, due to the dynamic programming like approach for vertices with low degrees, the complexity is low. The exact complexity depends on the degree of the vertices in the graphs, for more information see [**?**]. E.g. for the mapping of 10 processes to 16 possible processors, $16^{10} \approx 10^{12}$ solutions are possible, but the algorithm finishes within a few milliseconds. When the degree of the vertices increases, the computation time of the algorithm increases exponential. However, the process graphs are relative small (between 5 and 20 vertices) in our targeted application domain and in practice a process graph does not have a lot of vertices with degree ≥ 3. Therefore, we do not expect that the computation time will be a problem in practice.

Unfortunately, the algorithm does not take into account possible constraints. E.g. the capacity of processors and communication links are assumed to be infinite, which is not realistic. Also other constraints, e.g. restrictions on the delay in communication cannot be not taken into account by the MinWeight algorithm, which may be necessary to guarantee hard real-time behavior and Quality of Service (QoS). This is a serious limitation of the MinWeight algorithm.

C. Adding the Processor Capacity Constraint

The MinWeight algorithm can not handle additional constraints very well, e.g. the constraint that a processor has a limited capacity and therefore only a limited fix number of processes can run on a processor. Or even more advanced, it has to determine the number of processes that can run on a specific processor depending on the load of the processor in combination with the weight of the processes. To cope with the limited capacity of a processor, we adapted the MinWeight algorithm so that it satisfies the constraint that at most one process is mapped to each processor. A similar approach can be used for other constraints, e.g. at most two processes may be mapped to one processor. It is implemented in such a way that before computing the weight of a particular solution two conditions are checked. First, it checks whether the processors involved in the mapping solution are not already occupied in an earlier mapping step for another vertex of a processor graph. Second, it checks whether

Algorithm MINWEIGHT

(1) FOR $(u,v) \notin E_G$ and $p,q \in P$ DO $w_{pq}^{(u,v)} := 0$.
(2) Choose a vertex $v \in V_G$.
(3) IF $\deg(v) = 1$,
THEN let $V_G := V_G \backslash v$, and $E_G := E_G \backslash (u,v)$ for $u \in N(v)$.
FOR $u \in N(v)$ and $p \in P$ DO

$$w_p^u := w_p^u + \min_{q \in P}\{w_q^v + w_{pq}^{(u,v)}\}.$$

(4) IF $\deg(v) = 2$,
THEN let $V_G := V_G \backslash v$ and $E_G := (E_G \cup \{(x,z)\}) \backslash \{(x,v),(v,z)\}$ for $x,z \in N(v)$.
FOR $x,z \in N(v)$ and $p,q \in P$ DO

$$w_{pq}^{(x,z)} := w_{pq}^{(x,z)} + \min_{r \in P}\{w_r^v + w_{pr}^{(x,v)} + w_{rq}^{(v,z)}\}.$$

(5) IF $\deg(v) \geq 3$,
THEN choose a vertex $u \in N(v)$. Let $G := G/(u,v)$.
Set $P := P \times P$.
FOR $(p,q) \in P$ DO $w_{(p,q)}^{uv} := w_p^u + w_{pq}^{(u,v)} + w_q^v$.
FOR $x \in V_G$ and $(p,q) \in P$ DO $w_{(p,q)}^x := w_p^x$.
FOR $e = (uv,x)$ with $x \in N(uv)$ and $(p,q),(r,s) \in P$ DO $w_{(p,q)(r,s)}^{(uv,x)} := w_{pr}^{(u,x)} + w_{qr}^{(v,x)}$.
FOR $e \in E_G$ not incident with uv and $(p,q),(r,s) \in P$ DO $w_{(p,q)(r,s)}^e := w_{pr}^e$.
(6) IF $|V_G| \geq 2$, THEN GOTO (2).
(7) Output $w^* := \min\{w_p^u \mid p \in P, u \in V_G\}$. STOP.

Fig. 2. The MinWeight Algorithm

the processors involved in the mapping solution are all unique. Only if both conditions are satisfied, the solution is feasible.

The problem with adding these kind of constraints is that they introduces dependencies between the different assignment steps. Suppose we have two processes 1 and 2 and two processors A and B. Process 1 can be mapped to processor A and B and process 2 can only be mapped to processor B. Furthermore, processor B can execute only one process. If process 1 is mapped to process B, process 2 can not be assigned to processor B and therefore the mapping problem can not be solved in this way. In this example, the algorithm can even not find a solution. However, if a solution is found it is likely to be not an optimal one, because processes are competing for resources with limited capacity. One way to come up with an optimal solution is to iterate all the possible mappings. This solution is discussed in Section V. However, for practical use this is too time consuming and not possible at run-time. However, it is useful to know how far the adapted MinWeight solution deviates from the optimum.

D. Improvement of the Adapted MinWeight Algorithm

The adapted MinWeight problem suffers from two problems:

1) The algorithm does not find a mapping
   Different processes can compete for the same resources and it may happen that all the resources for a specific process are already occupied due to mapping decisions in the past. In this case, it is not possible for the algorithm to find a solution.
2) The algorithm finds a mapping that is far from optimal
   This is a result of the dependencies mentioned already.

The first problem is the most severe one. To reduce the chance of getting no feasible solution we may improve the MinWeight algorithm from Section IV-C by reordering the vertices that are chosen in step 2), see Figure 2. If a vertex only needs scarce resources, the probability is high that these resources are already taken by other processes when this vertex is mapped as one of the latest. Therefore, it is obvious that it is better to start with the mapping of vertices that needs only scarce resources to avoid resource bottlenecks instead of ending up with the result that the algorithm is not able to map the process graph to the SoC architecture.

When it is clear that there are no (longer) resource problems, it is better to start with mapping processes that have a high weight, because the quality of a solution is worse when a process with high processing costs is mapped inefficient, as when a process with low processing costs is mapped inefficient. Therefore, we propose an ordering of the vertices that is based on 1) the scarcity of the resources and 2) the weight of the processes.

However, it is not so simple to estimate when the resource scarcity is not longer a threat. It is important to detect as soon as possible that a reordering based on the weight of the processes is possible because this improves the optimality of the final solution. Currently, we are investigating which simple metric we may use to decide how to order the processes to obtain a possible near optimal mapping.

For the NoC in general we do not expect resource problems. Most tiled SoC architectures use a mesh structure for the NoC. That means that there are several different routes possible between two processing tiles with an equal length.

V. Quadratic Programming Solution

The assignment problem with the constraint that a processor may execute at most one process can be modeled as a quadratic assignment problem ([6], Section 12.9). This allows us to compute the optimal solution with the processor capacity constraint instead of a lower bound given by the MinWeight algorithm. We formulate our problem in a quadratic assignment problem as follows:

Input:

- a set P of p processors, $P = \{1, \ldots, p\}$.
- a weighted process graph $G_w = (V_G, E_G)$ with a weight vector w_p^u for all $u \in V_G$ and a weight matrix W^e for all $e \in E_G$ as defined in Section III-A.

Decision variable:

$$\text{let variable } x_{u,p} = \begin{cases} 1, & \text{if process } u \in V_G \\ & \text{is mapped on processor } p \in P \\ 0, & \text{otherwise} \end{cases}$$

Constraints:

$$\sum_{p \in P} x_{u,p} = 1,\ u \in V_G \text{(each process on exact 1 processor)}$$

$$\sum_{u \in V_G} x_{u,p} \leq 1,\ p \in P \text{(maximal 1 process per processor)}$$

$$x_{u,p} \in \{0,1\}, u \in V_G, p \in P$$

Objective function:

$$\min\left(\left(\sum_{u \in V_G}\sum_{p \in P} w_p^u \cdot x_{u,p}\right) + \left(\sum_{(u,v) \in E_G}\sum_{j \in P}\sum_{k \in P} x_{u,j} \cdot x_{v,k} \cdot w_{jk}^{(u,v)}\right)\right)$$

A. Pseudo code of Quadratic Problem Assignment Model

Existing algorithms (see [6]) can solve the quadratic assignment problem in a smart way, but for small instances also a simple brute force enumeration of all possible assignments is possible. In the following we give a pseudo code of such an algorithm to solve the Quadratic Problem Assignment problem in a brute force way. Two vectors are used:

- a vector `assign` with length n (number of processes), which denotes which process is assigned to which processor. For example, assign = [2,4,9] means that process 1 is assigned to processor 2, process 2 is assigned to processor 4 and process 3 is assigned to processor 9.
- a vector `busy` with length m (number of processors), which denotes whether a processor is already occupied with a process or not. For example, busy = [0,1,0] means that processor 1 is free and available for usage, processor 2 is busy with a process and processor 3 is free and available for usage.

Both vectors start at index 1 and are initialized with zeros. Furthermore, we assume the availability of two functions `nextfreeprocessor`, which return the next available processor or -1 when there is no next processor available and a function `evaluate`, which returns the costs of the current mapping. Figure 3 shows the pseudo code of the algorithm.

```
i = 1;
while (i > 0) {
  int np = nextfreeprocessor(..);
  if (np == -1) {
    busy [assign[i]] = 0;
    assign[i] = 0;
    i--;
  }
  else {
    busy[assign[i]] = 0;
    assign[i] = np;
    busy[np] = 1;
    if (i == n) { // leaf
      int costs = evaluate(..);
      if (costs < currentmin) {
        currentmin = costs;
        save solution
      }
    }
    else { // no leaf
      i++;
    }
  }
}
```

Fig. 3. Pseudo Code for Quadratic Assignment Algorithm

VI. Example

Digitale Radio Mondiale (DRM) [3] is a standard for digital radio below the 30 Mhz. A concise explanation of the DRM standard can be found in [4]. This section describes the mapping of a part of a DRM receiver to a heterogeneous tiled SoC architecture with 16 processing tiles. We use the template depicted in Figure 1. We number the tiles as follows: The left top tile is tile number 0, the direct right neighbor is tile number 1, the right bottom tile is tile number 15.

Figure 4 shows the processes of the digital baseband part of our DRM receiver. Table I shows the processes that we would like to map on the SoC (for a functional description of the processes see [4]). These processes concern the data flow of the DRM application; we do not consider the processes 9,10,11 in the "Global control & estimation" part of Figure 4. To test our algorithms, it is not crucial to have very accurate estimations of the weights. Therefore, we make a few assumptions to test our algorithms so that we do not have to realize the system to get the exact numbers:

- the number of multiplications per second is used as an indication for the costs of a process. Table I shows of the costs of the process in terms of multiplications per second for reception of Mode B, and the available implementations for the different type of processors.
- the ratio between processing an multiplication on an ASIC, DSRH, FPGA, DSP, GPP are 10:40:50:60:500 respectively.
- the communication costs increase linear with the distance of the communication path on the SoC. The communica-

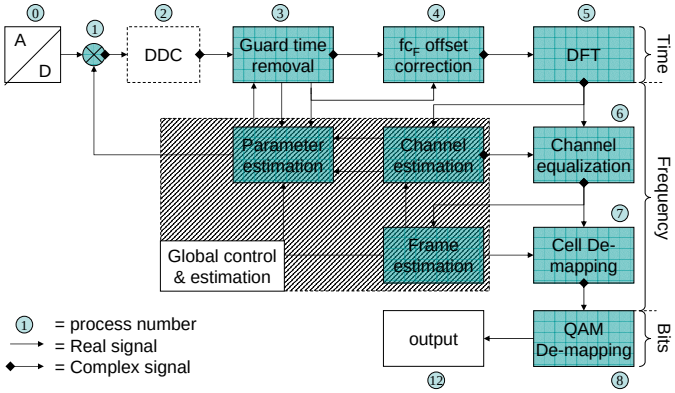

Fig. 4. DRM Processes to Map on SoC

tion costs are equal to the throughput in kbit/s given in Table II multiplied by the Manhattan distance between the tiles.

Note that processes that have an ASIC realization need a specific ASIC. It is not possible to assign a process with an ASIC realization to an arbitrary ASIC processor.

A. Results

Table III shows the solutions (the assignments and the total costs) of the different algorithms. In the mapping, position i gives the mapping of the ith process. So, e.g. for all mappings, process 3 (the fourth process) is assigned to processor 12.

The optimal mapping without constraints is given by the MinWeight algorithm. Note that the MinWeight algorithm maps different processes to the same DSRH tiles (6 and 13). If we assume that a tile may be used for at most one process there is a resource problem. Even by swapping some of these processes to other tiles of the same type, no feasible solution can be obtained, since 5 processes are assigned to the DSRH tiles, but only 4 instances of this type of tile are available.

Even when different tiles were chosen for this type of implementation of the processes, the mapping is not permitted if the DSRH may be used only for one process because this type of tile is 5 times requested and only 4 instances of this type of tile are available. Taking into account that every processor may run at most 1 process, another mapping is determined by the adapted MinWeight algorithm. Note that the initial processor mappings are the same and that a first difference occurs when tile 13 is used a second time. This gives a mapping that is 8% more expensive compared to the solution of the MinWeight algorithm. The remaining question is how much the solution of the adapted MinWeight differs from the optimal solution, which is expected to be higher than the lower bound given by the MinWeight algorithm due to the additional constraint. Therefore, the optimal solution is determined using the quadratic programming solution. It took several hours of computation on a Pentium 4 processor to evaluate all the possibilities with the brute force enumeration. This solution is about 3% more expensive than the MinWeight solution due to the additional processor capacity constraint. Therefore, we can conclude that we loose about 5% performance due to non optimality for the adapted MinWeight algorithm.

Block	Process	Multiplies	Processors
A/D converter	0	0	ASIC
Mixer	1	24k	DSP, DSRH, GPP
DDC	2	0	ASIC
Guard time correlation	3	144k	DSP, FPGA, GPP
Frequency Correction	4	96k	DSP, DSRH, GPP
FFT	5	346k	ASIC, DSP, DSRH, GPP
Channel equalization	6	38k	GPP, DSP, DSRH
Demapping	7	0	GPP, DSP, DSRH
Bit decoding	8	0	GPP, DSP, DSRH
Output	12	0	GPP

TABLE I

MULTIPLICATION COSTS FOR DRM MODE B

Edge	kbit/s
$0 \rightarrow 1$	375k
$1 \rightarrow 2$	750k
$2 \rightarrow 3$	755k
$3 \rightarrow 4$	600k
$4 \rightarrow 5$	600k
$5 \rightarrow 6$	300k
$6 \rightarrow 7$	241k
$7 \rightarrow 8$	201k
$8 \rightarrow 12$	78k

TABLE II

COMMUNICATION COSTS FOR DRM MODE B

algorithm	mapping	costs
MinWeight	5, 13, 9, 12, 13, 10, 6, 6, 6, 0	22324
Adapted MinWeight	5, 13, 9, 12, 6, 10, 7, 3, 2, 0	24188
Quadratic programming	5, 1, 9, 12, 13, 10, 6, 7, 11, 15	22985

TABLE III

DIFFERENT MAPPINGS

VII. CONCLUSION

The MinWeight algorithm computes very fast an optimal solution. However, the algorithm does not take into account all relevant constraints and therefore the practical use of the algorithm is limited. Adaptation of the MinWeight algorithm in order to fulfill the additional constraints gives a method which leads to a non-optimal solution. A realistic case shows that the adapted MinWeight algorithm gives a near optimal solution in a reasonable short computation time.

In future, we focus on three issues. First, additional constraints and heuristics will be added to the MinWeight algorithm to cope with more real life restrictions and to improve the solutions respectively. Second, we expect that adding heuristics to change the order in which the processes are mapped to processors improves the optimality of the solution. We are currently investigating how to determine a good ordering based on simple criteria. Third, another approach may be used so that in the

first step an optimal solution is computed using the MinWeight algorithm and in the second step the constraint violations are solved.

Acknowledgement

removed for the purpose of anonynous reviewing.

REFERENCES

[1] http://www.boost.org/libs/graph/doc/.

[2] P. Brucker. *Scheduling Algorithms*. Springer, 3 edition, 2001. ISBN:3-540-41510-6.

[3] E. T. S. I. (ETSI). Digital radio mondial (drm); system specification, Apr. 2003. ETSI ES 201 980 v1.2.2 (2003-04).

[4] F. Hofmann, C. Hansen, and W. Schäfer. Digital radio mondiale (drm) digital sound broadcasting in the AM bands. *IEEE Transactions on Broadcasting*, 49(3):319–328, Sept. 2003.

[5] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Proceeding of DATE04*, pages 234–239, Feb. 2004.

[6] W. L. Winston. *Operations Research; Applications and Algorithms*. International Thomson Publishing, 3 edition, 1993. ISBN: 0-534-20971-8.