J Supercomput (2012) 59:636–657
DOI 10.1007/s11227-010-0458-3

Efficient datapath merging for the overhead reduction of run-time reconfigurable systems

Mahmood Fazlali · Ali Zakerolhosseini · Georgi Gaydadjiev

Published online: 17 June 2010

Abstract High latencies in FPGA reconfiguration are known as a major overhead in run-time reconfigurable systems. This overhead can be reduced by merging multiple data flow graphs representing different kernels of the original program into a single (merged) datapath that will be configured less often compared to the separate datapaths scenario. However, the additional hardware introduced by this technique increases the kernels execution time. In this paper, we present a novel datapath merging technique that reduces both the configuration and execution times of kernels mapped on the reconfigurable fabric. Experimental results show up to 13% reduction in the configuration and execution times of kernels from media-bench workloads, compared to previous art on datapath merging. When compared to conventional high-level synthesis algorithms, our proposal reduces kernels configuration and execution times by up to 48%.

Keywords Reconfigurable computing · Run-time reconfigurable systems · Datapath merging

M. Fazlali (✉) · A. Zakerolhosseini
Department of Computer Engineering, Shahid Beheshti University G.C, Teheran, Iran
e-mail: fazlali@cc.sbu.ac.ir

M. Fazlali · G. Gaydadjiev
Computer Engineering Lab., Delft University of Technology, Delft, The Netherlands

A. Zakerolhosseini
e-mail: a-zaker@sbu.ac.ir

G. Gaydadjiev
e-mail: g.n.gaydadjiev@tudelft.nl

1 Introduction

Reconfigurable Computing (RC) systems rely on the efficient execution of computational intensive *kernels* using reconfigurable hardware. Current RC systems outperform conventional High-Performance Computers (HPC) due to their ability to customize the *Field Programmable Gate Array* (FPGA) resources according to the changing application requirements [1, 2]. Many scientific, multimedia, and communication applications have been reported in the literature as highly suitable for RC acceleration; especially in the HPC field, the number of industrial systems employing FPGA is steadily growing and their capabilities are constantly improving. In the vast majority of such systems, several kernels with high degree of parallelism are mapped to the FPGA as hardware accelerators (*modules*). The FPGA resources, however, are limited and often all modules cannot co-exist in the reconfigurable fabric at the same time. Hence, modules should be reconfigured on the FPGA at run-time to utilize the reconfigurable hardware during program execution. This process is referred as *Run-Time Reconfiguration* (*RTR*) [3].

Although RTR has the benefit of accelerating more modules than the reconfigurable fabric can fit at the same time, it introduces the reconfiguration time overhead. The time required for configuring the FPGA can potentially jeopardize the achieved acceleration in RTR systems. The configuration time can be efficiently hidden by using techniques such as configuration pre-fetching for applications where static schedule of the kernel loops is available in advance [4]. Such fully predictable applications, however, form only a small subset of the total class of applications suitable for RC acceleration. Therefore, it is important to consider techniques that reduce the configuration time overhead in a more generic scenario.

The dominating part of the configuration time is the time taken to load the new configuration bit-stream into the FPGA memory [4]. The configuration time is proportional to the length of the configuration bit-stream, and the speed of the configuration interface [5]. Reducing the length of the configuration bit-stream consequently reduces the module configuration time [6]. Previously, techniques such as configuration bit-stream compression have been proposed for reducing the configuration time. These techniques, however, introduce additional decompression penalties during the configuration process.

Another approach for reducing the configuration time is *High Level Synthesis* (*HLS*) [7, 8]. The basic idea used here is resource sharing by multiple Data Flow Graphs *DFG*[1] in order to minimize the configuration overhead. To this end, in [9], each DFG is synthesized by a special HLS algorithm that exploits intra-DFG resource sharing. In this way, multiple datapaths are produced for the execution of DFGs. Likewise, in datapath merging technique, two or more DFGs are merged based on inter-DFG resource sharing, in order to produce a single merged datapath [8]. The advantage comes from the fact that a single merged datapath effectively supports the functionality needed by two or more different datapaths and as a result the configuration time can be avoided. To quantify the efficiency of merged datapaths the *Module Aggregate Time* (T_A) that consists of both, the configuration time and the execution

[1]Data Flow Graph: is a graph which represents the data dependencies between the operations in a kernel.

time of the corresponding kernels is used. A trade-off exists between the merged datapath configuration time and the execution time of the kernels [10].

In the previous work mentioned above, T_A is not used as the optimization objective, neither the overall effect of the datapath configuration time and the kernel execution time in T_A reduction are considered.

Therefore, in this paper, we present a novel datapath merging technique, which reduces T_A by effective trade-off between the merged datapath configuration time and the execution time of the kernels. This is done by introducing *Minimal Aggregate Time Clique* (*MAT-Clique*) which is equivalent to the maximal clique problem [11]. Therefore, finding MAT-Clique is an *NP*-complete problem. We have addressed this by using a modified version of the Branch & Bound algorithm and as it will be shown in this paper; this technique can indeed reduce T_A.

The reminder of this paper is organized as follows. In Sect. 2, datapath merging is introduced and its influence on configuration time reduction is discussed. Relevant related work is presented and properly classified. Section 3 introduces the influence of datapath merging on T_A. The proposed datapath merging technique is explained in Sect. 4. The evaluation results are presented in Sect. 5, and finally Sect. 6 concludes the discussion.

2 Datapath merging and configuration time

The design flow used for hardware generation in RTR systems can be conceptually divided into two phases, namely, the synthesis and the configuration phase [9]. During the synthesis phase, suitable parts of an input program represented in a high-level programming language (i.e., C or Java) are translated to an intermediate representation. Each kernel is represented by its behavioral specification using a DFG that represents the order of basic operations and their interdependencies. Then all DFGs are transformed to a high-level Hardware Description Language (Verilog or VHDL) and synthesized using existing commercial tools that produce the actual bit-streams of the corresponding datapaths for the targeted reconfigurable device.

During the synthesis phase, resources needed by several DFGs can be reused (shared) to reduce the overall hardware cost. This process is performed off-line and creates coarser grain DFGs representing multiple kernels of the original program referred as *modules* in this paper. DFG merging algorithms can be used to target bit-stream size minimization.

In the second (configuration) phase, the bit-stream is loaded into the configuration memory of the reconfigurable device. Since in RTR systems this is performed at run-time, the bit-stream configuration time directly affects the RTR systems performance. Such configurations must be performed as efficiently and quickly as possible. This is to prevent the benefit gained by the hardware acceleration is eclipsed by the configuration time overhead [12].

2.1 Datapath merging

Assume that a DFG, $G = (V, E)$ corresponds to a kernel, where V is the set of vertices representing the basic operations and E is the set of edges defining the paths

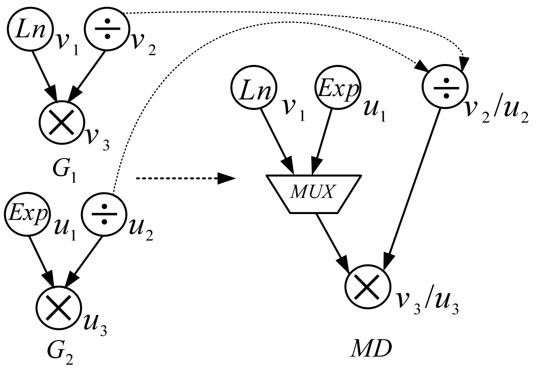

Fig. 1 Merging G_1 and G_2 to create *MD*

between operations. A vertex $v_i \in V$ contains basic operation performed by a hardware unit (e.g., specific functional unit, or registers to store variables) that has a set of input ports p. An edge $e = (v_i, v_j, p) \in E$, defines the data transfer from the vertex v_i to the input port p of vertex v_j. Vertex weight, $T(v_i)$ represents the hardware unit cost of a vertex $v_i \in V$.

A *merged datapath* $(MD) = (V', E')$, corresponding to two or more DFGs $G_i = (V_i, E_i)$, $i = 1 \ldots n$, is a directed graph. A vertex $v' \in V'$ represents merged vertices $v^j \in V_j$ and an edge $e' = (u', v', p') \in E'$ represents merged edges $e^j \in E_j$ where $j \in J \subseteq \{i, \ldots, n\}$.

Figure 1 depicts simple, motivating example of datapath merging where DFGs G_1 and G_2 are combined and *MD* is created. In this figure, $v_1 \in G_1$ and $u_1 \in G_2$ cannot be combined, consequently they remain in *MD* without any modification. A multiplexer is employed in the left input port of the vertex v_3/u_3 to select the correct input operand. $v_2 \in G_1$ and $u_2 \in G_2$ perform the same function (division) and can be mapped onto a single functional unit since the two DFGs are never coexisting at the same time. Then these vertices can be combined to create v_2/u_2 in *MD*. Similarly, $v_3 \in G_1$ and $u_3 \in G_2$ are merged into v_3/u_3. In this case, there is no need for a multiplexer in the right input port of $u_3/v_3 \in MD$ to select the input operands. Therefore, the edges v_2, v_3 and u_2, u_3 are merged and, the edge $(v_2/u_2, v_3/u_3)$ is created in *MD*.

The aim of datapath merging is to combine a number of DFGs in order to create a multimode datapath (module). This is an efficient method for reducing certain costs such as area usage, power consumption, or configuration time reduction in embedded systems and DSP applications [13–15].

On the other hand, resource sharing increases the hardware latency, and we have to consider trade-offs between hardware latencies and the implementation complexity [16]. In general, traditional HLS is not efficient for datapath merging in reconfigurable systems. However, combining DFGs in steps using maximum weighted clique can efficiently improve this [17, 18].

2.2 Related work

We can categorize previous works on configuration time reduction into two groups. The first group targets configuration time overhead at the synthesis phase, whereas the second group aims at reducing this overhead during the actual configuration.

In the first group, several works have been reported in the literature. A heuristic scheduling algorithm, which utilizes reconfigurable datapath components, has been presented in [19]. In this case, the resulting schedule is shortened so that the gain in execution clock cycles compensates the configuration time overhead. In [20], the execution order of the modules is rearranged to reduce the number of configurations, and hence minimize the configuration overhead. Also, [21] and [22] present algorithms to partition and schedule the modules and reduce the configuration time. In these algorithms, the configuration time overhead of each individual partition is considered. However, these approaches cannot be applied to many real applications where the kernel schedule is not known in advance. In [23], Boden et al. present an HLS technique targeting RTR systems which employs resource sharing to reduce the reusable design parts of the hardware implementation. Afterwards, in [24], they present GePaRD which is an approach to high level synthesis of self-adaptive systems, based on Partially Reconfigurable (PR) FPGAs. Although reducing reusable design parts decreases configuration time, this technique targets specific partitioned FPGA and is not generally applicable.

The second group focused on the optimization of the configuration process of reconfigurable units. Some researchers employed caching to decrease the configuration time [4]. However, spatial and temporal locality is not yet proven or widely accepted replacement policy principle for RTR systems. Many proposals for reducing the FPGA configuration time are based on shortening the bit-stream size. By doing so, compression methods have achieved suitable results for reducing the bit-stream length [12]. Although these methods are able to reduce the configuration time of RTR systems, performing them is costly due to the computational intensive decompression task. It is desirable that module's bit-stream's length is reduced at the system level (e.g., by HLS tools) while the hardware structures are generated in order to avoid these additional costs. Datapath merging has been shown as an effective HLS method for configuration time reduction [8]. Shannon and Diessel [25] show that graph merging can considerably reduce the configuration time. The merging process, however, introduces additional multiplexers in the merged datapath that will increase the hardware latency. Thus, by significant amount of kernel iterations, this may influence the overall execution time of a module. This disadvantage was considered in [10] where a method to create high-speed merged datapaths is proposed. This technique, however, cannot efficiently reduce T_A in RTR systems.

In this paper, we aim at reducing both, configuration and execution overheads in RTR systems by presenting a novel datapath merging method that efficiently minimizes T_A.

3 Module aggregate time (T_A) in datapath merging

T_A is the overall time, required to instantiate, and perform the kernels functionality implemented by a single module. In RTR systems, T_A consist of the *module configuration time* (T_C) and the *module execution time* (T_E). Therefore,

$$T_A = T_C + T_E. \tag{1}$$

Table 1 Abbreviations and notation

Abbreviation or notation	Explanation
T_e	Kernel execution time
T_{clk}	Maximum delay of the DFG stages
T_E	Module execution time
T_C	Module configuration time
T_A	Module aggregate time
ΔT_E	Reduction in module execution time
ΔT_C	Reduction in module configuration time
ΔT_A	Reduction in module aggregate time
T_F	Configuration time of hardware units
T_I	Configuration time of the interconnect
T_f	Functional unit configuration time (register)
T_{mux}	Configuration time of a multiplexer
W	Configuration time reduction
MD	Merged Datapath
MMD	Module-Aggregate-Time-Merged Datapath
G_c	Compatibility Graph
MAT-clique	Minimal Aggregate Time Clique
MAX-clique	Maximum Weighted Clique
Bound-clique	Bounded Execution Time Clique
C-HLS	Conventional HLS

The communication time of the module will change the absolute values of the experiment but will not impact the relative results. Therefore, for simplicity in our work, we do not consider the module communication time in T_A.

Datapath merging technique shortens T_C, but introduces additional multiplexers in the newly created *MD*, which on their turn will increase T_E. Kernels mapped to the reconfigurable hardware are usually heavily executed loops, therefore, any slight increase in a single iteration will consequently impact T_E (this is the total execution time of *all* iterations). This makes the number of kernel iterations important factor to consider in the merging method.

For sake of easy understanding, Table 1 summarizes the abbreviations and notations used in the rest of this paper.

Consider scheduled DFG with G corresponding to a kernel K, which can be executed in a number of time stages. As illustrated in (2), T_e is the kernel execution time, N is the number of iterations of K, and T_{clk} is the maximum delay of the separate time stages in G. By using pipelining for G, T_E for n kernels in a module will be the aggregate of all individual T_e. Therefore,

$$T_e = N T_{clk}, \qquad T_E = \sum_{i=1}^{n} T_{e_i} = \sum_{i=1}^{n} N_i T_{clk_i}. \tag{2}$$

Since RTR systems are used to accelerate the computational intensive kernels and those have significant number of iterations, we can safely ignore the time of prologue and epilog stages of the pipeline. This is the reason those times can be safely ignored in the above equation.

After merging DFGs G_i, $i = 1 \dots n$ in the module, a pipelined merged datapath MD is created. If the maximum delay of the MD pipeline stages (single clock cycle) is T'_{clk}, then T_e and T_E can be presented as

$$T_e = N T'_{clk}, \qquad T_E = \sum_{i=1}^{n} N_i T'_{clk_i}. \tag{3}$$

T_C is the required time to configure MD in the reconfigurable unit, that is,

$$T_C = T_F + T_I \tag{4}$$

where $T_F = \sum_{\forall v' \in V'} T_f(v')$ is the overall configuration time of the hardware units, and $T_I = \sum_{\forall v' \in V'} T_i(MUX)$ is the total configuration time of the MD interconnect. $T_f(v')$ is the configuration time of a functional unit or a register allocated to v', while $T_i(MUX)$ represents the configuration time of the additional multiplexers used at the input ports of each vertex v' [10].

3.1 Reduction of Module Aggregate Time (ΔT_A) in datapath merging

The main objective in our datapath merging algorithm is to reduce T_A. According to (1), T_A compromises T_C and T_E, while according to (4), T_C compromises T_F, and T_I. Therefore, to reduce T_A, all the components, i.e., T_F, T_I, and also T_E have to be simultaneously considered.

Figure 2 shows an example of merging DFGs that motivates our proposal. Each hardware unit and interconnect (the multiplexer) has its own configuration time. In Fig. 2(a), five simple DFGs, namely, $G_1 \dots G_5$ are illustrated. These DFGs are merged in consecutive steps, while T_C is reduced. As shown in Fig. 2(b), two DFGs G_1 and G_2 are combined and the merged datapath MD_1 is produced, while T_C is reduced.

In the next step, G_3 is merged to MD_1 to produce MD_2. In this case, a multiplexer is added to the input port of a vertex $a_2/b_2/c_2$ that will increase T_E. To merge a vertex $d_2 \in G_4$ onto a vertex $a_2/b_2/c_2 \in MD_2$, the increase in T_I (increase in multiplexer configuration time by using MUX4-1 instead of MUX2-1[2]) becomes bigger than the reduction in T_F. Therefore, in this step no vertex is merged onto MD_3, hence T_C is not reduced. In the same way, G_5 is merged onto MD_3 to reduce T_C and MD_4 is created.

Figure 2(c) illustrates two possible merged datapaths (MMD and MMD') resulting from the combination of the resources inside MD_4. It is clear that T_C of MMD and MMD' are shorter than the T_C of MD_4. T_C of MMD is shorter than the T_C of MMD'

[2]The multiplexors are 16 or 32 bits wide.

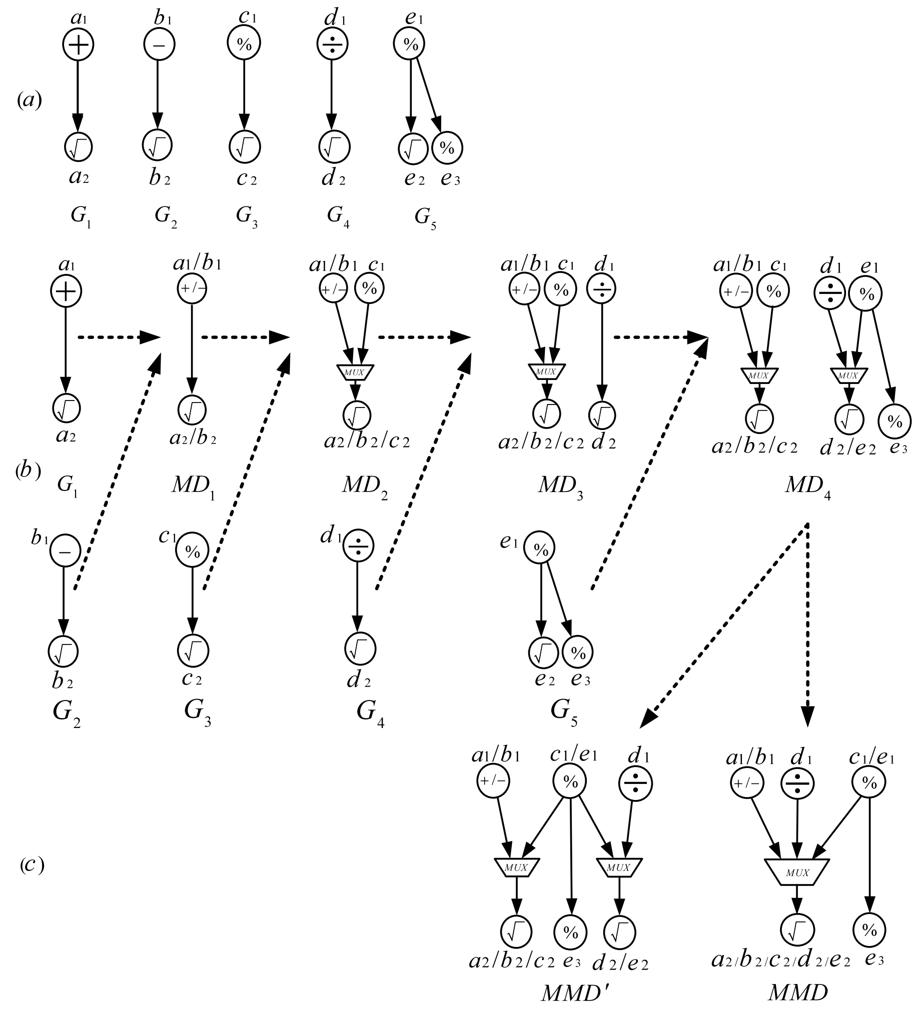

Fig. 2 (**a**) Five simple DFGs to merge, (**b**) merging DFGs in steps, (**c**) producing the merged datapath with least T_A

because two functional units (square root) have been merged in *MMD*. However, using *MUX*4-1 instead of *MUX*2-1 will impact T_E of *MMD* compared to *MMD*′. Therefore, T_A for the merged datapaths *MMD* and *MMD*′ heavily depends on the number of kernel iterations. It means that, when the number of kernel iterations increases, there is a trade-off between the two conflicting factors. More precisely, the module configuration time reduction (ΔT_C) and the increase in module execution time (ΔT_E) have to be balanced. This observation holds for every step of the datapath merging process.

From the above observations, we conclude that if we merge DFGs only when considering the hardware unit configuration time (T_F), the interconnect configuration time (T_I), the kernel execution time (T_e), and T_A can be efficiently reduced. Therefore, in this paper, we present a new approach referred to as *Module-Aggregate-Time-Merged Datapath* (*MMD*).

MMD is a merged datapath with minimal T_A to instantiate a module and execute the kernels implemented in it.

The rest of the section formalizes ΔT_C and ΔT_E, resulting from merging DFGs, to explain our datapath merging technique.

After merging vertices v^j, $j = 1 \ldots m$ from DFGs G_i, $i = 1 \ldots n$, a vertex v' is created in the *MMD*. Moreover, for each input port of v' which has more than one incoming edge, a multiplexer is added to the *MMD* for selecting the correct input operand.

Thus, the *configuration time reduction* (w) for the merging process is

$$w = \sum_j T_f\big(v^j\big) - \big(T_f(v') + Tmux_{\mathrm{p}}\big). \tag{5}$$

In this equation, $T_f(v^j)$ and $T_f(v')$ represent the T_f for a vertex $v^j \in G_j$ and for a vertex $v' \in MMD$ respectively, while $Tmux_{\mathrm{p}}$ is the multiplexer configuration time. According to this equation, w is proportional to T_f and $Tmux_{\mathrm{p}}$.

After merging the edges, the number of multiplexers at the input ports of vertex v' is decreased [18]. Therefore, the multiplexer's size is reduced and the $Tmux_{\mathrm{p}}$ drops. Hence, w obtained by merging the edges is equal to the reduction in multiplexer configuration time ($\Delta Tmux_{\mathrm{p}}$), for k number of ports. That is,

$$w = k \times \Delta Tmux_{\mathrm{p}} \tag{6}$$

ΔT_C after merging vertices and edges from different DFGs is equal to the aggregate of w as result of the merging vertices and edges. That is,

$$\Delta T_C = \sum w. \tag{7}$$

Now, if we designate the increase in T_E as ΔT_E then

$$\Delta T_E = \sum_{i=1}^{n} N_i T'_{clk_i} - \sum_{i=1}^{n} N_i T_{clk_i} = \sum_{i=1}^{n} N_i \Delta T_{clk_i}. \tag{8}$$

In (8), ΔT_{clk} represents the increase in the delay of the time stage after merging DFGs. We define the reduction in T_A as ΔT_A which is given as

$$\Delta T_A = \Delta T_C - \Delta T_E. \tag{9}$$

For a significant number of kernel iterations N, the trade-off between the two conflicting factors ΔT_C and ΔT_E has to be solved. According to (9), in order to reduce T_A, we should maximize ΔT_A. Thus, the problem can be defined as:

Given a module with n DFGs G_i, $i = 1 \ldots n$, corresponding to different kernels, merge all G_i to create MMD, such that ΔT_A is maximal.

To efficiently reduce T_A, we maximize ΔT_A during the datapath merging sequences (see the example in Fig. 2), while ΔT_E and ΔT_C are also considered. In the next section, we describe our proposal in more detail.

4 The proposed datapath merging method

At run-time, during program execution a number of modules are to be configured and executed on the reconfigurable hardware. Since the separate kernels in a module will be executed consecutively, we base our approach on the fact that only one of the implemented DFGs will be active at a given time. Hence, during datapath merging, resources across DFGs can be shared without introducing resource conflicts. Based on the trade-off between ΔT_C and ΔT_E, we can implement *MMD* for arbitrary number of kernel iterations, such that a number of different implementations of *MMD* are made available. The operating system can select at run-time the most suitable implementation based on the system dynamic state [26]. Therefore, we have to reduce T_A for each implementation of the *MMD*.

Merging all DFGs at the same time is an *NP*-hard problem [17]. The researcher in [27] used Linear Programming (*LP*) algorithm to solve the problem. However, an *LP* algorithm does not converge where the number of nodes in DFGs increases [8, 27]. Therefore, we use a greedy algorithm somehow similar to the one described in [8]. This is implemented by using the compatibility graph that returns the merged datapath close to an optimal solution.

Hence, in the datapath merging technique proposed in this paper, DFGs G_i, $i = 1 \ldots n$ from the module are combined in several steps. For each step of datapath merging, excluding the final step, DFGs are merged together one by one. At each step, G_i is merged onto the *MD* while ΔT_A is maximized. During the final stage, the resources in the last *MD* are combined together thus creating *MMD*.

For all steps of the datapath merging, except of the final step, all merging possibilities for combining vertices and edges between *MD* and the G_i are considered. Then in the final step, all merging possibilities inside *MD* are also considered and *MMD* is created. The *compatibility graph* represents the merging possibilities. Note that the compatibility graph used here resembles the one used by the merging method as described in [18].

Now, we define compatibility graph, $G_c = (N, A)$, as an undirected weighted graph where,

- Each node $n \in N$ corresponds to:

  Merging a vertex $v_j \in G$ and a vertex $v_j \in MD$ to create a vertex (v'), in order to modify *MD* or, merging an edge $e_i \in G$ and another edge $e_j \in MD$ to create the edge $e' = (u', v', p')$ in order to modify *MD*. This is applicable to all steps, excluding the final step.

  Merging vertices $v_i, v_j, \ldots, v_k \in MD$ to create a vertex v' in *MMD*, or merging edges $e_i, e_j, \ldots, e_k \in MD$ to create an edge e' in *MMD*, where it does not merge two vertices from a DFG together. This is applicable to the final step only.

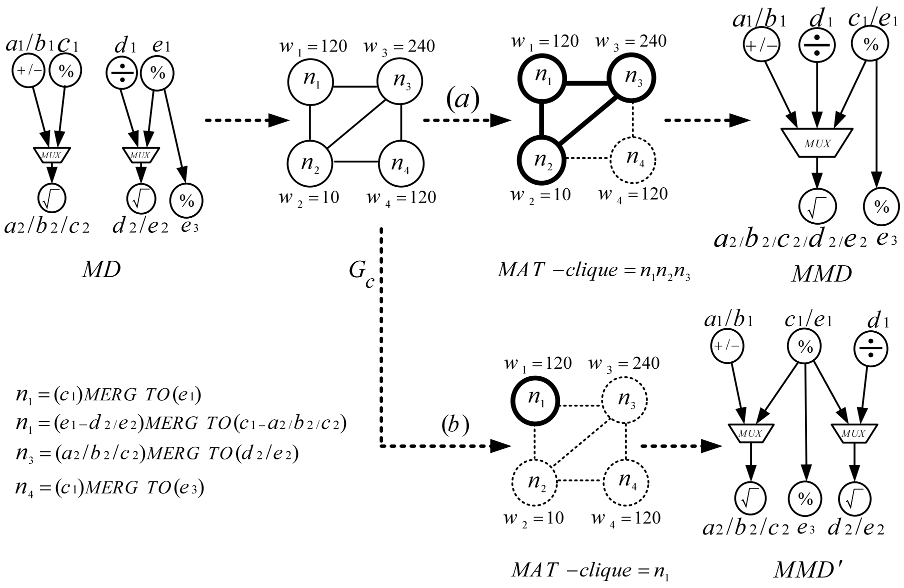
a_1/b_1c_1
d_1 e_1
$a_2/b_2/c_2$ d_2/e_2 e_3
MD
$w_1=120$ $w_3=240$
n_1 n_3 n_2 n_4
$w_2=10$ $w_4=120$
G_c
(a)
MAT-clique $= n_1n_2n_3$
a_1/b_1 d_1 c_1/e_1
MUX
$a_2/b_2/c_2/d_2/e_2$ e_3
MMD
$n_1=(c_1)$MERG TO(e_1)
$n_1=(e_1\text{-}d_2/e_2)$MERG TO$(c_1\text{-}a_2/b_2/c_2)$
$n_3=(a_2/b_2/c_2)$MERG TO(d_2/e_2)
$n_4=(c_1)$MERG TO(e_3)
(b)
MAT-clique $= n_1$
a_1/b_1 c_1/e_1 d_1
$a_2/b_2/c_2$ e_3 d_2/e_2
MMD'

Fig. 3 G_c is the compatibility graph for input *MD*. (**a**) Creating *MMD* where $\Delta T_E = 0$. (**b**) Creating *MMD* where $\Delta T_E \ll \Delta T_C$

- Each arc, $a = (n, m) \in G_c$, illustrates that the nodes n and m are compatible.[3]
- The nodes weight, w, represents the configuration time reduction, as described in Sect. 3.1.

In Fig. 3, G_c illustrates the compatibility graph for merging vertices inside *MD*. Consider two nodes $n_1 = (c_1, e_1)$ and $n_4 = (c_1, e_3)$, chosen together from G_c for datapath merging. This will result in merging a vertex $c_1 \in G_3$ onto two vertices $e_1 \in G_5$ and $e_3 \in G_5$. Therefore, these nodes are not compatible and there is no arc between n_1 and n_4 in G_c.

Next, when G_c has been created, we determine the number of compatible nodes in G_c (paired adjacent nodes) that maximizes ΔT_A. This is done by defining *Minimal Aggregate Time Clique* (*MAT-Clique*). In brief, MAT-clique is a clique in G_c that has minimal T_A for m kernels corresponding to G, $i = 1, \ldots, m$.

4.1 Minimal Aggregate Time Clique (MAT-Clique)

Choosing compatible nodes from $G_c = (N, A)$, is the process of searching a completely connected the subgraph in G_c, which is called *clique* in the Graph theory [11]. Clique weight is the total weights of all clique nodes. In our case, the node weights in G_c are the configuration time reductions (w). Consequently, the weight of the clique is equal to ΔT_C, resulting from the datapath merging corresponding to

[3]In the compatibility graph $G_c = (N, A)$, two nodes $n = (v_1, u_1) \in G_c$ and $m = (v_2, u_2) \in G_c$ are compatible while $v_1 \neq v_2$ if and only if $u_1 \neq u_2$. This means that the merges indicated by the two nodes of G_c are compatible if they do not merge two vertices from a DFG together.

the clique. The maximum weighted clique (MAX-Clique) is a clique in G_c where the total weight of its nodes is larger than any other clique in G_c. Therefore, weight of MAX-clique gives the maximum achievable ΔT_C. It means using MAX-clique to merge DFGs maximizes ΔT_C at each step of the datapath merging, and creates a *MD* with minimal T_C [8].

However, in this paper, we are going to reduce T_A, thus we maximize ΔT_A in each step of the datapath merging process, namely, by searching for a clique in G_c that maximizes ΔT_A.

Definition 1 Given a compatibility graph G_c for DFGs, G_i, $i = 1, \ldots, m$, at each step of the datapath merging process, the **MAT-clique** is a clique in G_c that has minimal T_A for m kernels corresponding to G, $i = 1, \ldots, m$.

According to (9), and assuming that a kernel has a single iteration, the T_E will be negligible compared to T_C, thus $\Delta T_A = \Delta T_C$ and MAT-clique = Max-clique. Therefore, our problem is equal to determining the Max-clique in G_c. After merging the resources in *MD* as shown in Fig. 3, which corresponds to merging G_i, $i = 1, \ldots, n$, two *MMD* can be created based on the number of kernel iterations. In this way, two pairs of vertices and two edges from *MD* have been merged together in order to create *MMD*. Figure 3(a) shows the *MMD* whenever the kernel loops have single iteration. In this case, determining the MAT-clique from the compatibility graph G_c, is equal to determining the MAX-clique from G_c and *MMD* is created using this clique.

When the number of kernel iterations is significant, we must calculate ΔT_E and subtract it from ΔT_C to obtain ΔT_A at each stage of our datapath merging process. Figure 3(b) represents the case when the numbers of kernel iterations increases. MAT-clique is different from MAX-clique in this case. By using this clique, vertices c_1 and e_1 from *MD* have been merged to create a vertex c_1/e_1 in *MMD*$'$. In order to avoid using *MUX4-1* (which results in T_E increase), Fig. 3(b) presents a scenario where the vertices $a_2/b_2/c_2$ and d_2/e_2 are not merged. Therefore, determining the MAT-clique is similar to searching for MAX-clique. Thus, we have changed the algorithm of MAX-clique to determine the desired MAT-clique.

Determining the MAX-clique is a well-known problem in graph algorithms. It is proved to be an *NP*-Complete problem [28]. In order to find a solution, the Branch and Bound algorithm can be employed [29]. The Branch and Bound algorithm chooses an efficient method to select nodes and predicts bounds for quick backtracking. It assumes only positive weights for the nodes of the input graph.

However, in the G_c of the proposed technique, although some nodes corresponding to the merged vertices may have negative weights, the nodes corresponding to the merged edges have only positive weights and the final result will be positive. This implies that, in our proposal, the nodes of G_c have signed integers as weights.

The proposed algorithm for finding the MAT-Clique is an improved version of the algorithm in [29] and uses similar optimizations as indicated in their algorithm to reduce the problem search space. More precisely, we have made the following modifications to the algorithm presented in [29]:

1. Our algorithm adds the negative weight vertices to the clique at the end of each branch to support the signed integer weight for the nodes. This way, we subtract ΔT_E from ΔT_C within each merging step and ΔT_A is calculated.

```
$S_i$ ={$n_i$, $n_{i+1}$,... $n_n$}/* set of nodes from i to n in $G_c$ */
U ={$n_1$, $n_2$,... $n_i$}/* set of remained nodes from 1 to i in $G_c$ in each branch */
N($n_i$) /* a set of adjacent nodes to node $n_i$ in $G_c$*/
$M_t$ /* MAT- Clique*/
MaximumWeight /* a variable for the weights of MAT- Cliques*/
Weight[i] /* array of the nodes' weights in $G_c$ */
W /* Weight of the clique in each branch */
$W_e$ /*Increase in execution time*/
AddNegativeWeightNodes(clique)/*function to add negative nodes to a clique*/
C[i] /* array to save the weights of the cliques in ForLoop*/
Sort(clique) /* a function to sort the nodes in the input graph*/
Clique /* a variable to store the clique in each branch */
-------------------------------------------------------------------
Function MAT-Clique (input: Graph $G_c$ ($N_c$,$A_c$),DFGs $G_i$ i=1..n, output: Clique $M_t$ =($N_t$,$A_t$))
Begin
1 Maxweight =0;
2 S ← Sort ($G_c$);
3 For i←n downto 1 do /*calling function Cliquer for all nodes in $G_c$*/
4   Cliquer ($S_i \cap N(n_i)$, Weight[i]-$W_e$)
5   C[i] ← Maxweight;
6 EndFor;
7 return $M_t$;
End MAT-Clique

Procedure Cliquer (Input: U, $W_c$)
Begin
8  If (|U|==0) then /* checking the end of branch*/
9    AddNegativeWeightNodes(Clique);
10   If (W > Maxweight) then
11       Maxweight← W;
12       $M_t$ ← Clique;
13   End If
14   return
15 End If
16 While (|U|>0)do
17   i ←smallest position of remaining nodes in U;
18   If($W + \sum_{vi \in U} weight[i] - MAX(W_e)) < MaxWeight$) Then /* The first Bound */
19     return;
20   If ((W + C[i]) < MaxWeight) /* The second Bound */
21     return;
22   U ← {U - $n_i$};
23   Clique ← {Clique + $n_i$};
24   Cliquer($U \cap N(n_i)$, W + Weight[i]- $W_e$);
25 End While
26 return;
End Cliquer
```

Fig. 4 Pseudocode of the proposed algorithm to determine MAT-Clique

2. The algorithm applies two bounds (see Fig. 4) for T_A to the proposed Branch & Bound algorithm to reduce the problem space.

```
AggregateTimeDatapathMerging (Input: DFGs $G_i$=($V_i$,$E_i$), $i$=1...n, Output: MMD=(V,E))
  {assuming $G_i$ i=1...n are sorted based on their size};

Begin
/*First stage of merging: Calling the DPM function to produce MD for $G_i$ i=1...n */
  MD ←DPM($G_i$=($V_i$,$E_i$) i=1...n);
/*Second stage of merging: Finding Intra-Merged Datapath Resource Sharing */
 $G_c$ ←Create-Compatibility-Graph(MD);
 $M_t$ ←MAT-Clique($G_c$, $G_i$ i=1..n);
 MDD ←Create-Merged-Datapath($M_t$,MD);
End AggregateTimeDatapathMerging
-----------------------------------------------------------------------
DPM (Input: DFGs $G_i$=($V_i$,$E_i$) i=1...n,Output: merged datapath MD=(V,E))
Begin
/*Finding Inter-DFG Resource sharing */
 MD←$G_1$;
 for i←2 to n do
    $G_c$ ←Create-Compatibility-Graph(MD,$G_i$);
    $M_t$ ← MAT-Clique ($G_c$, $G_i$ i=1..n);
    MD ←Reconstruct-Merged-Datapath($M_t$,MD,$G_i$);
  Endfor
 End DPM
```

Fig. 5 The proposed algorithm to determine *MMD*

The proposed *MAT-Clique* function, shown in Fig. 4, determines the MAT-Clique in the input graph G_c. At the beginning, the function sorts all nodes of G_c based on the nodes' weight and a sorted set S is created that contains all nodes. By using continue growing subsets of S incrementally, from the smallest subset to the biggest one, the recursive function *Cliquer* is called for each subset of S. Each time the *Cliquer* is called, it searches all the cliques in G_c, which might be the MAT-Clique.

The bounds are used to cut the branches and limit the problem space to obtain the MAT-Clique. It searches the branches to find a clique which leads us to the maximal ΔT_A. The obtained clique of each branch is compared to M_t and the one which maximizes ΔT_A is saved in M_t. When the main loop is completed, the MAT-clique will be in M_t.

4.2 The proposed datapath merging algorithm

We merge DFGs in steps. The *MAT-Clique* function presented above is employed to search the nodes and edges in G_c, which should be merged in each step. Below, the pseudocode of the proposed datapath merging algorithm is shown in Fig. 5. In the first step, the datapath merging function in Fig. 5 is called. In this function, the DFGs are merged one by one in a loop. The First DFG is considered as *MD*. In each loop iteration, G_c between *MD* and the next DFG, G_i, is created by using the function *Create-Compatibility-Graph*. This function is based on the first definition of the G_c that shows all merging possibilities between the vertices (or edges) from G_j and the vertices (or edges) from *MD*. Then the function *MAT-Clique* creates the *MAT-Clique* in G_c and stores it in M_t.

The function *Reconstruct-Merged-Datapath* uses M_t to reconstruct *MD* and to modify *MD* for the next step of the loop. This way, M_t is used to add the vertices and edges of the G_i onto *MD* and reconstruct it. These steps are repeated for merging all DFGs to *MD*. The *MD* obtained by *DPM* function is the result of the first stage to merge DFGs.

In the second stage, G_c is created by using the function *Create-Compatibility-Graph*. G_c is created based on the second definition of compatibility graph, which shows the merging possibilities among the vertices and edges inside the *MD*. After that, the function *MAT-Clique* creates *MAT-Clique* in G_c and stores it in M_t. At the end, M_t is used to merge the vertices and the edges in *MD* to produce the *MMD*.

In each step of the proposed datapath merging method where $\Delta T_C \le \Delta T_E$ (that means $\Delta T_A \le 0$), there is not a *MAT-Clique* in that step of the algorithm. So, the function *MAT-Clique* returns a null value to avoid increase in T_A. Using the proposed algorithm, the DFGs corresponding to kernels in the module are merged and the desired merged datapath with minimal T_A will be produced.

5 Experimental results

In order to investigate the effectiveness of our technique and its associated high level synthesis algorithm, we performed experiments using some well-known workloads from the media-bench suite [30]. Enough experimental evidence exists to support the fact that there are some kernels (computational intensive loops) in each program of the media-bench, which have the largest contribution to the benchmark execution time [17]. The characteristic of the kernels makes them suitable for mapping on reconfigurable unit in RTR system. Each program was compiled using the *GCC* compiler [31], and was profiled to determine which kernels contributed most to the overall program execution time.

For each such kernel, a DFG was generated from the loop body RTL code (GCC intermediate representation). Using RTL instead of machine instructions permitted us to extract the loop code after most machine-independent code optimizations, but before register allocation and the machine-dependent transformations. Moreover, whenever possible, procedure integration (automatic in-lining) was applied. The section of the application code corresponding to a DFG can contain control constructions, such as "if-then," "if-then-else," and "switch." For simplicity, we did not consider nested loops. The DFGs were generated using a technique based on if-conversion and using condition bit vectors. We considered up to three kernels/DFGs for each of the applications. The resource usage of the control unit in datapath merging is lower than the control unit in conventional synthesis [20]. Therefore, the control units for the input DFGs were not taken into account on the investigated merged datapaths.

The vertex's weights in DFGs are representing the configuration times. So, we have to calculate hardware unit configuration time (T_f) and multiplexer configuration time T_{mux} to use in the input DFGs in the synthesis techniques and compare the results of the techniques with each other. After obtaining the bit-streams of hardware units and multiplexers using ISE 10.2, their configuration times were approximated as: configuration time = [(size of bit-stream)/(FPGA Virtex5-xcv5vlx configuration clock frequency)] [5]. The configuration clock frequency 100 MHz was used, in our experiment.

To evaluate the results of the proposed datapath merging method, we implemented three previous HLS techniques, which have been proposed for reduction of the configuration overhead in RTR systems. These techniques were compared to the proposed method, with respect to the reduction in T_A in the targeted programs. The former technique is based on the method presented in [9], which uses a conventional HLS technique for intra-DFG resource sharing in RTR systems. The second one is a datapath merging technique in [8] that searches inter-DFG resource sharing for datapath configuration time reduction in RTR systems. The third technique is the datapath merging technique in [10] for the overhead reduction in RTR systems that aims at high-speed merged datapaths.

The first technique in [9] uses conventional HLS and will be referred to as *C-HLS*. The second technique in [8] uses the Maximum weighted clique approach to merge DFGs and is referred to as *Max-Clique*. The third technique in [10] which uses Bounded execution time Clique to make high speed merged datapath is referred as *Bounded-Clique*. Our approach, which is based on computing the MAT-Clique is referred to as *MAT-Clique*.

To implement the C-HLS, the DFGs were scheduled and the mobility of the vertices was calculated. Then the resources in the DFG were shared using the Integer Linear Programming (ILP) method to create the pipelined datapaths. To apply datapath merging techniques to the DFGs, initially the DFGs were scheduled and chaining was exploited during scheduling. Later, the algorithms of Max-Clique, Bounded-Clique, and the MAT-Clique were applied to the DFGs to create the corresponding pipelined merged datapaths.

We applied the above techniques to three input DFGs corresponding to the kernels in the Mpeg2-decoder benchmark. There are several possible combination forms for the DFGs for each iteration of the kernel loops. We considered the same number of iterations (N) for all kernel loops in the module in each experiment to compare the results. For each module, the datapaths resulting from applying the conventional HLS to DFGs were introduced. Then datapath configuration time was estimated as: configuration time = [(size of datapath bit-stream)/(100 MHz)]. Similarly, the merged datapaths resulting from applying the Max-Clique, Bounded-Clique, and MAT-Clique were created and their module configuration times (T_C) were estimated. Module execution times (T_E) was achieved based on the N and clock cycle of the pipelined datapaths for C-HLS, and clock cycle of the pipelined merged datapaths for datapath merging techniques. Similarly, module aggregate time (T_A) was calculated based on T_C and T_E. Table 2 presents the results obtained after applying these techniques to the DFGs of the Mpeg2-decoder program for the various N. In this table T_C, T_E and T_A are reported for all the techniques where $N = 10$ up to $N = 50000$.

As illustrated in this table, for $N = 10$, T_E is much smaller than T_C in all techniques. Therefore, in this case, T_C is the biggest portion of T_A. Besides this, T_C in the MAT-Clique is much shorter than the T_C in the C-HLS, and shorter than T_C in the Max-Clique and the Bounded-Clique. As a result, the MAT-Clique had the least T_A compared to the rest.

By increasing N, T_C is steady in C-HLS and the Max-Clique techniques. However, T_C increases in the Bounded-Clique and the MAT-Clique techniques. The reason is the trade-off between reduction in module configuration time (ΔT_C) and increase in module execution time (ΔT_E). Nonetheless, the small increase in T_C in

Table 2 T_C, T_E, and T_A, in Mpeg2-decoder program for the algorithms in case of FPGA Virtex5-xcv5vlx

Kernel loop iterations (N)	C-HLS			Max-Clique			Bounded-Clique			MAT- Clique		
	T_C (ms)	T_E (ms)	T_A (ms)	T_C (ms)	T_E (ms)	T_A (ms)	T_C (ms)	T_E (ms)	T_A (ms)	T_C (ms)	T_E (ms)	T_A (ms)
10	5.83666	0.00020	5.83686	4.11511	0.00023	4.11534	4.11511	0.00023	4.11534	3.64192	0.00024	3.64216
10000	5.83666	1.69626	7.53292	4.11511	2.39309	6.5082	4.11511	2.39309	6.5082	4.08183	2.39309	6.47492
20000	5.83666	3.09251	8.92917	4.11511	4.78617	8.90128	5.82322	3.12226	8.94548	5.22322	3.51902	8.74224
30000	5.83666	5.08877	10.9255	4.11511	7.17926	11.2944	5.82322	5.35643	11.1797	5.34331	5.09683	10.4401
40000	5.83666	6.78503	12.6217	4.11511	9.57235	13.6875	5.82322	7.49537	13.3186	5.38629	6.92156	12.3079
50000	5.83666	8.48129	14.3179	4.11511	11.9654	16.0805	5.82322	9.98698	15.8102	5.50664	8.75683	14.2635

these techniques prevents increase in T_E, which results in better T_A for the Bounded-Clique and the MAT-Clique techniques. Although for $N = 20000$, Bounded-Clique creates high-speed merged datapath, which has the shortest T_E of all techniques, T_A in Bounded-Clique is longer than T_A resulting from Max-Clique. However, in this case, MAT-Clique creates a merged datapath with the minimal T_A. For $N > 30000$, T_A in the Max-Clique and Bounded-Clique is larger than T_A in C-HLS. However, due to the trade-off between ΔT_C and ΔT_E, MAT-Clique performs better than others.

We performed similar experiment for Epic-Decoder and Epic-Encoder programs from the Media-bench benchmark suite. Each program contains a module consisting of three DFGs. For each module, ΔT_A was obtained. We calculated the reduction percentage in T_A after merging DFGs by the techniques. The data in the Fig. 6 is obtained by comparing the T_A and software execution time for Mpeg2-Decoder, Epic-Decoder and Epic-Encoder programs. Figure 6a illustrates the reduction percentage in T_A for Mpeg2-decoder program. It shows that generally by increasing N, the reduction rate of T_A is slowed down. As illustrated in Fig. 6a, the MAT-Clique shows the biggest reduction percentage in T_A for all values of N. For $N > 30000$, reduction in T_A in the C-HLS is more than the reduction in T_A in the Max-Clique and Bounded-Clique techniques. For $N > 40000$, these techniques perform unsatisfactory an increase T_A. The reason is the usage of the multiplexers on the critical path delay of the merged datapaths that increases T_E considerably.

Figure 6b shows the reduction percentage in T_A for Epic-Decoder module. As shown in this figure, the C-HLS has not reduced T_A considerably compared to other techniques (the reduction percentage in T_A is close to zero), while the MAT-Clique shortens T_A more than other techniques. As shown in Fig. 6c, the behavior of the techniques for Epic-Encoder program is similar to their behavior for Epic-Decoder and Mpeg2-decoder programs, in that MAT-Clique performs better than other techniques. For $N > 30000$, MAT-Clique and C-HLS have not reduced T_A, while Max-Clique and Bounded-Clique have increased T_A.

Merging DFGs in steps, using the compatibility graph, is the fastest solution for efficient datapath merging [17]. Therefore, we compared the synthesis time of the proposed Branch and Bound algorithm (MAT-Clique) with the Max-Clique and Bounded-Clique techniques that both also merge DFGs in steps. Figure 7 shows the synthesis times of these techniques after being applied to the DFGs of the studied applications. The DFGs in Empeg2-encoder and G721 modules have higher number of vertices compared to the DFGs in Epic-decoder, Epic-encoder and Empeg2-decoder modules.

As illustrated in Fig. 7, the synthesis time of MAT-Clique technique for Epic-decoder, Epic-encoder, and Empeg2-decoder modules is longer than the synthesis time of Max-Clique and Bounded-Clique algorithms. Also, the synthesis time of MAT-Clique for Empeg2-encoder and G721 applications is shorter than the synthesis time of Max-Clique and Bounded-Clique techniques. This means that where the number of vertices in DFGs increases, adding the bound (for T_A) to our Branch and Bound algorithm can reduce the synthesis time compared to the other two algorithms. Since datapath merging is an *NP*-complete problem, the presented results are valuable and more significant when merging DFGs of considerable sizes.

The results show that the proposed datapath merging algorithm is a valid HLS method for RTR systems. It significantly reduces T_C compared to conventional syn-

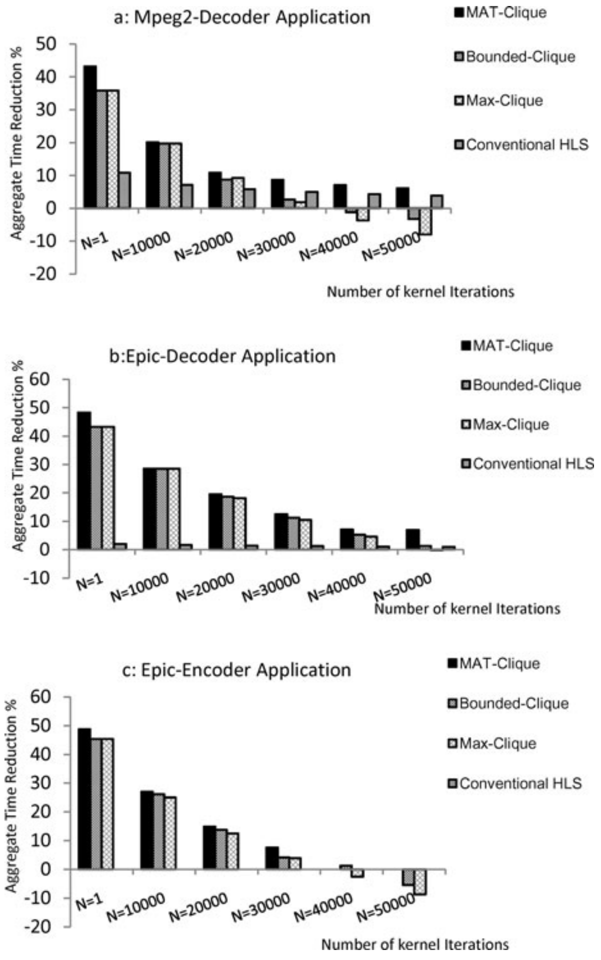

Fig. 6 The reduction percentage in T_A in Mpeg2-Decoder, Epic-Decoder, and Epic-Encoder modules for the algorithms in case of FPGA Virtex5-xcv5vlx

thesis algorithms and also is better than any previously proposed datapath merging algorithms. On the other hand, while previously used datapath merging techniques do not reduce T_A for different number of kernel iterations, the proposed datapath merging technique has successfully performed this task.

6 Conclusions

This paper presented an efficient datapath merging technique for the reduction of T_A (the sum of configuration and overall execution times) of two or more repetitive kernels in RTR systems. The kernels were represented as DFGs with a number of iterations and merged together to create a merged datapath. We considered the trade-off between merged datapath configuration time and the execution times of the corresponding kernel loops to reduce T_A. The similarity between the DFGs was represented as a compatibility graph. Afterward, we mapped our merging solution to the

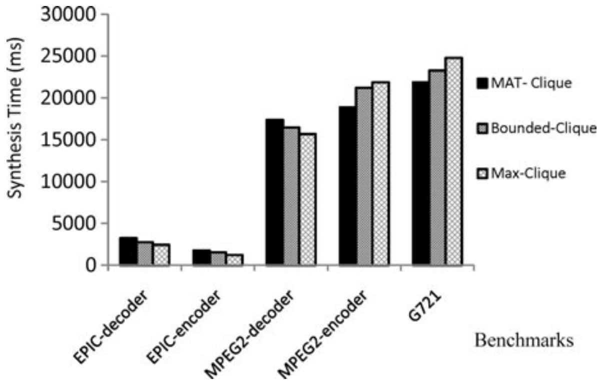

Fig. 7 Synthesis time of the MAT-Clique, Bounded-Clique and the Max-Clique algorithms to merge DFGs

problem of determining the MAT-Clique from the compatibility graph, which is a *NP*-complete. We presented a modified version of the Branch and Bound algorithm to solve this problem in polynomial time. To evaluate the proposed datapath merging technique, we applied it to the applications from media-bench suite where Virtex5-xcv5vlx FPGA device was our target. The results indicate that the proposed datapath merging technique outperforms the conventional synthesis and all previously proposed datapath merging techniques aimed at RTR systems. Our experiments show that T_A has been reduced by up to 48% compared to the C-HLS techniques, and up to 13% compared to the previous datapath merging techniques. Moreover, the synthesis time of the proposed datapath merging technique is improved compared to the previous datapath merging methods, where the DFGs contains significant amount of nodes. For programs where the module scheduling is completely predictable and is available beforehand, the configuration time can be effectively hidden by well-known techniques such as prefetching and will not benefit from the proposed method. Such programs, however, form a tiny subset of the complete application space and are not very often used.

Acknowledgements The authors would like to thank the Iran Telecommunication Research Center (ITRC) and Hartes project for supporting this research in the context of the project T/500/3462 and project EU-IST-035143, respectively.

References

1. Woods N (2007) Integrating FPGAs in high-performance computing: the architecture and implementation perspective. In: Fifteenth ACM/SIGDA international symposium on field-programmable gate arrays (FPGA), pp 132–137
2. El-Ghazawi T, El-Araby E, Huang M, Gaj K, Kindratenko V, Buell D (2008) The promise of high-performance reconfigurable computing. Computer 41(2):69–76
3. Compton K, Hauck S (2002) Reconfigurable computing: a survey of systems and software. ACM Comput Surv 34(2):171–210
4. Li Z (2002) Configuration management techniques for reconfigurable computing. PhD thesis Northwestern University
5. Rollmann M, Merker R (2008) A cost model for partial dynamic reconfiguration. In: International conference on embedded computer systems: architectures, modeling, and simulation (SAMOS). July 2008, Samos, Greece, pp 182–186

6. Huang Z, Malik S (2001) Managing dynamic reconfiguration overhead in systems-on-a-chip design using reconfigurable datapaths and optimized interconnection networks. In: Proceeding of design automation and test in Europe (DATE), March 2001, Munich, Germany, pp 735–740
7. Coussy Ph, Morawiec A (2008) High-level synthesis from algorithm to digital circuit. Springer, Berlin
8. Fazlali M, Zakerolhosseini A, Sabeghi M, Bertels K, Gaydadjiev G (2009) Datapath configuration time reduction for run-time reconfigurable systems. In: International conference on engineering of reconfigurable systems and algorithms (ERSA), July 2009, Nevada, USA, pp 323–327
9. Qu Y, Tiensyrj K, Soininen J-P, Nurmi J (2008) Design flow instantiation for run-time reconfigurable systems. EURASIP J Embed Syst 2(11):1–9
10. Fazlali M, Zakerolhosseini A, Sahhbahrami A, Gaydadjiev G (2009) High speed merged datapath design for run-time reconfigurable systems. In: International conference on field-programmable technology (FPT), December 2009, Sydney, Australia, pp 339–342
11. Kumlander D (2001) A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In: European congress of mathematics (4ECM), June–July 2001, Stockholm, Sweden, pp 202–208
12. Farshadjam F, Dehghan M, Fathy M, Ahmadi M (2006) A new compression based approach for reconfiguration overhead reduction in virtex based RTR systems. Comput Electr Eng 32(4):322–347
13. Chavet C, Andriamisaina C, Coussy Ph, Casseau E, Juin E, Urard P, Martin E (2007) A design flow dedicated to multimode architectures for DSP applications. In: International conference on computer-aided design (ICCAD), November 2007, San Jose, CA, USA, pp 604–611
14. Boden M, Fiebig T, Meißner T, Rülke S, Becker JA (2007) High-level synthesis of HW tasks targeting run-time reconfigurable FPGAs. In: IEEE international symposium on parallel and distributed processing (IPDPS 2007), March 2007, CA, USA, pp 1–8
15. Chiou L, Bhunia S, Roy K (2005) Synthesis of application-specific highly efficient multi-mode cores for embedded systems. ACM Trans Embed Syst Comput 4(1):168–188
16. Zuluaga M, Topham N (2008) Resource sharing in custom instruction set extensions. In: Symposium on application specific processors (SASP), June 2008, Wellington, DC, USA, pp 7–13
17. Moreano N, Borin E, de Souza C, Araujo G (2005) Efficient datapath merging for partially reconfigurable architectures. IEEE Trans Comput-Aided Des 24(7):969–980
18. Fazlali M Fallah KF, Zolghadr M, Zakerolhosseini A (2009) A new datapath merging method for reconfigurable systems. In: 5th international workshop on applied reconfigurable computing (ARC), March 2009, Karlsruhe, Germany, pp 157–168
19. Economakos G (2006) High-level synthesis with reconfigurable datapath components. In: IEEE international conference on parallel and distributed processing symposium (IPDPS), April 2006, Rhodes Island, Greece
20. Ghiasi S, Nahapetian A, Sarrafzadeh M (2004) An optimal algorithm for minimizing run-time reconfiguration delay. ACM Trans Embed Comput Syst 3(2):237–256
21. Mehdipour F, Saheb-Zamani M, Ahmadifar HR, Sedighi M, Murakami K (2006) Reducing reconfiguration time of reconfigurable computing systems in integrated temporal partitioning and physical design framework. In: IEEE international parallel and distributed processing symposium (IPDPS), April 2006, Rhodes Island, Greece, pp 219–230
22. Cordone R, Redaelli F, Redaelli MA, Santambrogio MD, Sciuto D (2009) Partitioning and scheduling of task graphs on partially dynamically reconfigurable FPGA. IEEE Trans Comput-Aided Des Integr Circuits Syst 28(5):662–675
23. Brisk P, Kaplan A, Sarrafzadeh M (2004) Area-efficient instruction set synthesis for reconfigurable system-on-chip designs. In: Annual conference on design automation (DAC), June 2004, San Diego, CA, USA, pp 395–400
24. Boden M, Fiebig T, Reiband M, Reichel P, Rulke S (2008) GePaRD a high-level generation flow for partially reconfigurable designs. In: IEEE computer society annual symposium on VLSI (ISVLSI), April 2008, France, pp 298–303
25. Shannon K, Diessel O (2007) Module graph merging and placement to reduce reconfiguration overheads in paged FPGA devices. In: international conference on field programmable logic and applications (FPL), August 2007, Amsterdam, Netherlands, pp 293–298
26. Fu W, Compton K (2005) An execution environment for reconfigurable computing. In: Annual IEEE symposium on field-programmable custom computing machines (FCCM), April 2005, CA, USA, pp 149–158
27. de Souza C, Lima AM, Moreano N, Araujo G (2005) The datapath merging problem in reconfigurable systems: Lower bounds and heuristic evaluation. ACM J Exp Algorithmic 10(2):1

28. Garey M, Johnson DS (1979) Computers and intractability-a guide to the theory of NP-completeness. Freeman, San Francisco
29. Ostergard PRJ (2002) A fast algorithm for the maximum-weighted clique problem. Discrete Appl Math 120(1–3):197–207
30. Lee C, Potkonjak M, Mangione WS (1997) Media-bench: a tool for evaluating and synthesizing multimedia and communication systems. In: Annual IEEE/ACM international symposium on micro-architecture (MICRO), December 1997, California, USA, pp 330–335
31. GNU compiler collection internals. http://gcc.gnu.org/onlinedocs