Energy-Efficient Mapping of Real-Time Streaming Applications on Cluster Heterogeneous MPSoCs

Di Liu*, Jelena Spasic*, Gang Chen†, Todor Stefanov*
*Leiden University, †Technical University of Munich
Email: {d.liu, j.spasic, t.p.stefanov}@liacs.leidenuniv.nl, cheng@in.tum.de

***Abstract*—In this paper, we propose a novel polynomial time algorithm, called Frequency Driven Mapping, to map real-time streaming applications specified as cyclo-static dataflow (CSDF) graphs onto a cluster heterogeneous MPSoC. The objective of our mapping approach is to reduce the energy consumption and guarantee latency and throughput constraints. The main novelty in our mapping algorithm is twofold: 1) By using hard-real-time scheduling of CSDF graphs, we propose an efficient way to determine a suitable processor type for each task in a CSDF graph, where the energy consumption is minimized and throughput and latency constraints are met; 2) According to an initial mapping derived by a first-fit-decreasing heuristic, we propose a remapping approach, where some tasks are remapped to unused clusters in order to further reduce the energy consumption of the system by cluster dynamic voltage/frequency scaling (DVFS). The experimental results show that the proposed algorithm finds more energy efficient mapping compared to existing approaches. The energy savings due to our proposed algorithm are up to 34%.**

I. Introduction

We have been in the multiprocessor era for many years, where multiprocessor systems span from supercomputers to embedded systems. In the embedded system domain, multiple processors and other components are integrated into one chip and this is widely known as Multiprocessor System-on-Chip (MPSoC). The growth of MPSoC enables embedded systems to run a plethora of complex applications, especially streaming applications, which have high computational demands, e.g., video and audio streaming, video conferencing, etc. However, the need for efficiently utilizing MPSoC resources brings several challenges to system designers.

How to utilize the parallelism available in MPSoCs and how to efficiently map the complex software to the hardware of current MPSoCs are the two major challenges. To address the former challenge, several Models of Computation (MoCs) have been proposed to parallelize applications running on an MPSoC, e.g., Kahn Process Network (KPN) [1], Synchronous Dataflow (SDF) [2], and Cyclo-Static Dataflow (CSDF) [3], such that applications can efficiently exploit the parallelism available in MPSoCs. For the latter challenge, a significant amount of work has been done to map the software to the hardware considering different constraints, hardware architectures, and objectives [4], e.g., throughput, latency, energy, etc.

Among these objectives, energy efficiency now has been deemed as one of the main objectives for MPSoC system design. To achieve the energy efficiency, dynamic voltage/frequency scaling (DVFS) is a widely used technique to save energy, where each processor is able to change the voltage and frequency to reduce power consumption. However, with the advent of manycore systems, per-core DVFS becomes impractical due to the high hardware cost and area requirement [5]. In order to balance the energy saving and hardware cost, cluster MPSoCs or MPSoCs with several voltage frequency islands (VFI) emerge as a solution. On cluster MPSoCs and VFI MPSoCs, all processors on the same cluster or VFI operate at the same voltage and frequency level. Two examples of a cluster MPSoC and VFI MPSoC are ARM big.LITTLE [6] which has two clusters and Intel SCC [7] which has 24 frequency islands and 6 voltage islands. In addition, the cluster MPSoCs, e.g., ARM big.LITTLE [6], also feature single-ISA heterogeneous cores/processors [8]. Due to the distinguished power-performance property of different types of processors, single-ISA heterogeneous MPSoCs expose more opportunities to save energy, while the same ISA provides an easy way for tasks to migrate with low overhead [6]. Throughout this paper, when we refer to heterogeneous MPSoCs, we mean single-ISA heterogeneous MPSoC systems.

Although the cluster heterogeneous MPSoC concept, described above, has shown its energy efficiency in the state-of-the-art chips, e.g., Samsung Exynos 5422 [9], there has been no sufficient effort by the design community to devise a systematic approach for mapping real-time streaming applications onto a cluster heterogeneous MPSoC. Thus, motivated by this fact, in this paper we propose a novel algorithm to efficiently map real-time streaming applications onto cluster heterogeneous MPSoCs, which are subject to latency and throughput constraints, such that the energy consumption of the cluster heterogeneous MPSoC can be reduced by using cluster DVFS. The proposed algorithm mainly consists of three phases: 1) The first phase determines the processor type for each application task such that the energy consumption can be reduced by utilizing the available heterogeneity in the MPSoC, while guaranteeing the performance constraints, i.e., throughput and latency; 2) Based on the processor type assignment in the first phase, the second phase maps tasks to clusters. Then by using cluster DVFS, the energy consumption of the system can be reduced; 3) Based on the second phase, the third phase remaps tasks to unused clusters in order to further reduce the energy consumption.

In this paper, we have the following contributions:

- We propose a novel polynomial time algorithm, called Frequency Driven Mapping (FDM), to map real-time streaming applications onto a cluster heterogeneous MPSoC with the aim of reducing the energy consumption and guaranteeing the latency and throughput constraints. The main novelty in this algorithm is twofold: 1) By using the hard-real-time scheduling of CSDF graphs, explained in Section IV-B, we propose an efficient way to determine a suitable processor type for each actor/task in the CSDF graph, where the energy consumption is minimized and throughput and latency constraints are met (the first phase mentioned above); 2) According to an initial mapping derived by the first-fit-decreasing (FFD) heuristic (the second phase mentioned above) and the properties of cluster MPSoCs, we remap some tasks to unused clusters in order to further reduce the energy consumption (the third phase).
- We performed various experiments on real-life streaming applications. The experimental results show that compared

978-1-4673-8164-2/15/$31.00

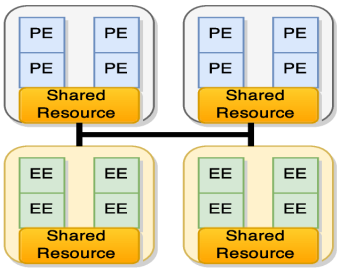

Fig. 1: An example of a cluster heterogeneous MPSoC

to the existing approaches in [10] and [11], the proposed FDM algorithm can achieve more energy saving.

The remainder of the paper is organized as follows. Section II describes the assumptions considered in our work. Section III discusses the related work. Section IV introduces the models used in our work. Section V presents the proposed mapping algorithm. The experimental results are presented in Section VI, and Section VII concludes the paper.

II. Scope of Work

In this section, we explain the assumptions considered in our paper. We consider a cluster heterogeneous MPSoC which has two types of clusters, namely performance-efficient (PE) clusters and energy-efficient (EE) clusters, which refer to the ARM's **big.LITTLE** architecture [6]. Each cluster has a number of identical processors, PE processor or EE processor. In total, the cluster heterogeneous MPSoC consists of $N_c^{\text{PE}} \times N_p^{\text{PE}}$ PE processors and $N_c^{\text{EE}} \times N_p^{\text{EE}}$ EE processors, where N_c^{PE} and N_p^{PE} represent the total number of PE clusters and the number of processors per PE cluster, respectively. N_c^{EE} and N_p^{EE} are the total number of EE clusters and the number of processors per EE cluster, respectively. Fig. 1 shows an example of a cluster heterogeneous MPSoC, where there are two PE clusters and two EE clusters, each cluster with four identical processors. The processors within one cluster share some resources, e.g, the last level cache, the memory controller, etc. A cluster in the system can be switched off, thereby consuming no power. In this paper, we only consider symmetric clusters, i.e., all clusters have the same number of processors, but our approach can be easily adapted to asymmetric clusters.

For streaming applications, to effectively exploit the parallelism available in an application, we model the streaming applications as acyclic CSDF graphs. The authors in [12] have shown empirically that 80% of the streaming applications can be modeled as acyclic graphs. But, instead of scheduling CSDF-modeled streaming applications by using conventional dataflow scheduling, we apply the hard-real-time (HRT) scheduling on CSDF-modeled streaming applications explained in Section IV-B. Scheduling a CSDF graph in this HRT fashion makes it possible to benefit from a large amount of proven theories developed by the real-time systems community, which provide several efficient and fast approaches to test the schedulability of an application when mapped onto a given MPSoC.

By using the HRT scheduling approach mentioned above, we can deploy various real-time scheduling algorithms to schedule the tasks of a CSDF graph. Hard-real-time scheduling algorithms are mainly divided into two categories, partitioned scheduling and global scheduling algorithms [13]. Partitioned scheduling means that tasks are statically assigned to a processor and restricted to run on the assigned processor, and then uniprocessor schedulability tests can be used to check the schedulability. However, the partitioned scheduling is analogous to a bin-packing problem which is known to be an NP-hard problem [14]. Moreover, partitioned algorithms are known to be non-optimal for scheduling real-time tasks [13]. On the other side, global scheduling holds a global task queue for a system and assigns tasks to processors at runtime according to the availability of processors, and task migration is allowed to efficiently utilize the system resources. The global scheduling provides a flexible way to schedule real-time tasks and several global scheduling algorithms are proven to be optimal for real-time multiprocessor systems, e.g., PFair [15] and LLREF [16] algorithms. The main concerns for global scheduling are the cache coherence and the task migration overhead, but ARM big.LITTLE already provides hardware support for low-overhead task migration which makes the global scheduling more practical.[1]

However, since it is extremely complex and costly to have fully global scheduling with DVFS which might need a per-task DVFS or a complex run-time monitor for each processor in order to meet deadlines, a cluster scheduling is used in this paper to schedule tasks, where tasks are assigned statically to a cluster and globally scheduled within the cluster. The cluster scheduling provides a good trade-off between the global scheduling and the partitioned scheduling with respect to schedulability and scalability [17]. In this paper, we assume that on each cluster an optimal global scheduling algorithm, e.g., PFair [15] or LLREF [16], is deployed to schedule tasks/actors. Considering cluster scheduling, we can compute the minimum frequency for each cluster offline such that the system can reduce its energy consumption.

III. Related Work

The energy-efficient design issue has been addressed extensively in the past decade. Dynamic frequency/voltage scaling (DVFS) and dynamic power management (DPM) are the two major techniques used to achieve energy efficiency. In particular, in real-time systems, a considerable amount of work has been done to deploy DVFS to schedule real-time tasks in an energy-efficient way. [18] surveys most of the papers dealing with energy efficient scheduling of real-time tasks by using DVFS. All papers reported in [18] assume per-core DVFS platforms. Regardless of the properties of cluster MPSoCs, directly adopting these per-core DVFS approaches onto cluster MPSoCs may lead to an energy inefficient design/mapping [11]. In contrast, the algorithm proposed in this paper is specifically devised for cluster heterogeneous MPSoCs and effectively deals with the mapping problem on the cluster heterogeneous MPSoCs in terms of energy efficiency.

Although the works in [10] and [11] have addressed real-time task mapping on cluster-based MPSoC systems, they are different from our work in several aspects. The differences are summarized in Table I. First, [10] and [11] only consider to meet deadlines for each task, but do not consider the applications' performance constraints, e.g., throughput and latency, which streaming applications usually are subject to. Applying their techniques directly onto the considered streaming applications will not guarantee the performance constraints, i.e., throughput and latency. Second, they consider partitioned scheduling in their work, whereas motivated by the state-of-the-art hardware, i.e., ARM big.LITTLE, we consider a cluster scheduling which provides a good trade-off between global scheduling and partitioned scheduling with respect to schedulability and scalability [17]. Thus, the cluster scheduling can achieve a better system utilization than partitioned scheduling. Finally, the work in [11] only considers homogeneous MPSoCs, whereas we consider heterogeneous MPSoCs which provide better energy efficiency.

Some works have been done to reduce the energy consumption of streaming applications on MPSoCs [19], [20], [21] and [22], where some performance metrics are taken into account.

[1] CoreLink CCI-400 Cache coherence interconnect [6]

TABLE I: The difference from [11] and [10]

	heterogeneous	performance	scheduling
Colin et al. [10]	Yes	No	Partitioned
Kong et al. [11]	No	No	Partitioned
Ours	Yes	Yes	Cluster

The works in [19], [20], [21] and [22]use per-core DVFS on homogeneous MPSoCs to reduce energy consumption. In contrast, we consider cluster heterogeneous MPSoCs in our work. Heterogeneous MPSoCs are known to be more energy efficient than homogeneous MPSoCs [8]. Since finding a suitable type of processor for each actor/task is really important with respect to energy reduction and guaranteed performance, mapping tasks to heterogeneous MPSoCs is a more challenging job than mapping tasks to homogeneous MPSoCs. Moreover, adopting the per-core DVFS approaches without considering the properties of cluster MPSoCs may result in an energy inefficient mapping [11].

IV. Preliminaries

In this section, we first provide an overview of the CSDF model. Then, the hard-real-time scheduling of the CSDF model is introduced. After that, the system model and energy model used in this paper are given.

A. Cyclo-Static Dataflow (CSDF)

A CSDF graph is defined as a directed graph $G = (V, E)$, where V is a set of actors and E is a set of edges. Actor $\tau_i \in V$ represents computation and edges represent the transfer of data tokens between actors. Each actor $\tau_i \in V$ may consume/produce a varied but predefined number of data tokens in its consequent executions, called *consumption/production sequence*.

It has been proven in [3] that a valid static schedule of a CSDF graph can be generated at design-time if the graph is consistent and live. A CSDF graph is said to be consistent if a non-trivial solution exists for the *repetition vector* $\vec{q} = [q_1, q_2, \ldots, q_i]$. An entry q_i indicates the number of invocations of actor τ_i in one graph iteration of G. For more details, we refer the reader to [3].

B. Hard-Real-Time (HRT) Scheduling of CSDF

It has been shown in [23] that an acyclic CSDF graph can be converted to a periodic taskset, if the *worst-case execution time* (WCET) C_i of each actor is known. Consequently, a plethora of well-developed real-time theories, e.g., scheduling theories and schedulability tests, can be applied to schedule or analyze the CSDF graph or to replace the complex design space exploration for homogeneous MPSoC design [24].

To schedule actors of a CSDF graph G in HRT fashion, the period of each actor needs to be computed according to its *repetition value* and *worst-case execution time (WCET)*. In this paper, **the overhead due to data communication among actors is included in the WCETs**. In order to deal with the variance of different mappings, we take the worst-case communication overhead into account such that the feasibility of our approach can be guaranteed.

To better understand the concept of computing the actors' periods, we give the following definition:

Definition 1. *The **workload** of an actor τ_i is $W_i = q_i C_i$ and the **maximum actor workload** of the graph is $\hat{W} = \max_{\tau_i \in G}\{W_i\}$*

As a result, the minimum period $\breve{T}_i$ of actor τ_i can be computed by the following equation [23]:

$$\breve{T}_i = \frac{\text{lcm}(\vec{q})}{q_i} \lceil \frac{\hat{W}}{\text{lcm}(\vec{q})} \rceil \tag{1}$$

where $\text{lcm}(\vec{q})$ is the *least common multiple* of the *repetition vector* $\vec{q}$ (explained in Sectiontion IV-A). Here, the deadline of each actor is set to be equal to its period, called *implicit deadline periodic task* (IDP) in the real-time theory. In addition, the conversion procedure of a CSDF graph to periodic tasks computes the start time S_i for each actor/task as well. Due to the space limitation, we do not show the formula to compute S_i. Start time S_i of actor τ_i is determined by the deadline of its predecessor which finishes its execution last and the data dependency between actor τ_i and its predecessors. The reader is referred to [23] for the full formulae and explanation. With the given and computed parameters mentioned above, actor τ_i is characterized by a tuple $\tau_i = \{C_i, S_i, \breve{T}_i\}$.

After the periodic tasks are derived, the latency and throughput of the CSDF graph scheduled in the HRT fashion can be computed. Eq. (2) is used to compute the minimum latency of the CSDF graph scheduled in HRT fashion.

$$L(G) = \max_{w_{in\to out} \in \mathcal{W}} (S_{\text{out}} + (g^C_{\text{out}} + 1)\breve{T}_{\text{out}} - (S_{\text{in}} + g^P_{\text{in}}\breve{T}_{\text{in}})) \tag{2}$$

where $w_{in\to out}$ is one path of set $\mathcal{W}$ which consists of all paths from the input actor to the output actor. Here, S_{out} and $\breve{T}_{\text{out}}$ are the start time and period, respectively, of output actor τ_{out}, while S_{in} and $\breve{T}_{\text{in}}$ denote the start time and period, respectively, of input actor τ_{in}. g^C_{out} and g^P_{in} are two constants which denote the number of invocations the actor waits for the non-zero consumption/production of tokens on a path $w_{in\to out} \in \mathcal{W}$. The throughput of the CSDF graph is computed as follows:

$$\mathcal{R} = 1/\breve{T}_{\text{out}} \tag{3}$$

Note that when all actors have the minimum periods, the graph can reach the minimum latency and the maximum throughput achievable by the HRT scheduling. In this paper, we take these minimum achievable latency and maximum achievable throughput as the performance constraints. *Note that we can set other throughput and latency constraints by using the period scaling technique [25] on the minimum periods computed by Eq. (1).*

C. System Model

Since actors may run on different processor types, WCET C_i varies according to the performance of the assigned processor type. Hence, we first extend the task model used in Section IV-B to support this WCET variation. We replace the scalar C_i with a vector $\vec{C_i}$ which consists of two WCETs, C_i^{EE} and C_i^{PE}, corresponding to the WCET of a task running on an EE processor and on a PE processor, respectively. C_i^{EE} and C_i^{PE} are the WCETs when EE and PE processors run at their maximum operating frequencies supported by the hardware platform.

As we mentioned in Section II, we adopt cluster scheduling in this paper. With cluster scheduling, we compute the minimum voltage/frequency level offline and configure the cluster with the computed voltage/frequency level. On each cluster, an optimal global scheduling algorithm, e.g., PFair [15] or LLREF [16], is deployed to schedule actors. A periodic taskset is schedulable on a homogeneous multiprocessor system by an optimal global scheduling algorithm, if $U = \sum_{\forall \tau_i \in V} C_i/T_i \leq M$ [26], where U is the total utilization of the taskset and M is the number of processors. Since in our work we consider an optimal global scheduling algorithm for each cluster, tasks mapped onto a PE cluster are schedulable if $U_k^{PE} \leq N_p^{\text{PE}}$, while tasks mapped onto an EE cluster are schedulable if $U_j^{EE} \leq N_p^{\text{EE}}$. U_k^{PE} and U_j^{EE} are the utilization of PE cluster k and EE cluster j, respectively. They are computed by the following equations:

$$U_j^{\text{EE}} = \sum_{\forall \tau_i \in V_j^{\text{EE}}} \frac{C_i^{\text{EE}}}{\breve{T}_i}, \qquad U_k^{\text{PE}} = \sum_{\forall \tau_i \in V_k^{\text{PE}}} \frac{C_i^{\text{PE}}}{\breve{T}_i} \tag{4}$$

where V_j^{EE} and V_k^{PE} represent actors assigned to EE cluster j and PE cluster k, respectively.

TABLE II: The 'uncore' power consumption

f (GHz)	2	1.8	1.6	1.4	1.2	1	0.8
$P_s^{\text{PE}}(f)$ (W)	0.8	0.528	0.39	0.309	0.244	0.182	0.134
f (GHz)	1.4	1.2	1	0.8	0.6	0.4	0.2
$P_s^{\text{EE}}(f)$ (W)	0.04	0.04	0.04	0.04	0.04	0.04	0.04

D. Energy Model

In this paper, we use the real measurements from the ODROID XU-3 [27] board to build our power and energy models. The ODROID XU-3 has an Exynos 5422 chip [9], where there are two clusters on the chip, one quad core Cortex A15 (big) and one quad core Cortex A7 (little). The power consumption of a cluster consists of two parts, 'processor' and 'uncore' [28]. The 'processor' power consumption is power dissipated by the processors, while the 'uncore' power consumption is the power consumption from some components not pertaining to a processor, e.g., a shared cache, an integrated memory controller, etc. The 'uncore' power consumption of the ODROID XU-3 is shown in Table II, where the 'uncore' power consumption for the PE (big) clusters and EE (little) cluster, at different operating frequencies, are given. We can see that for the PE (big) cluster the 'uncore' power consumption $P_s^{\text{PE}}(f)$ scales along with the cluster operating frequency. We find that the 'uncore' power consumption $P_s^{\text{PE}}(f)$ contributes approximately 20% to the total power consumption of the big cluster. For the EE (little) cluster, with the on-chip power sensor, we can not see the variation of the 'uncore' power consumption $P_s^{\text{EE}}(f)$ at different frequency levels. Hence, in Table II, the 'uncore' power consumption $P_s^{\text{EE}}(f)$ is the same for each frequency level. Note that since one core on the little cluster has to be active for the operating system, we are not able to measure the pure 'uncore' power consumption $P_s^{\text{EE}}(f)$ for the little cluster. The values of $P_s^{\text{EE}}(f)$ given in Table II include some power consumption from the active core running the OS.

Although the 'uncore' power consumption may be related to the frequency as shown in Table II, it is different from the dynamic power consumption which also relates to the frequency. Dynamic power is only consumed when there is a workload on the processors, whereas the 'uncore' power consumption always exists as long as the cluster is on. Based on the above discussion, we use the following power model for each cluster,

$$P(f) = \alpha f^b + N_p\beta + P_s(f) \tag{5}$$

where the first term is the dynamic power consumption, β is the static power consumption of one processor and N_p is the number of processors on the cluster. $P_s(f)$ is the 'uncore' power consumption and f is the frequency level. In this paper, we use power parameters from ODROID XU-3 as our reference, so parameters α, b, and β are estimated by using curve fitting with real power measurements from the ODROID XU-3 board. The estimated parameters for each processor type are shown in Table III.

TABLE III: The estimated parameters

processor type	α (W/MHz^b)	b	β (W)
PE (big)	3.03×10^{-9}	2.621	0.155
EE (little)	2.62×10^{-9}	2.12	0.0278

To validate our power model, described above, we measure the power consumption from the board and compare it with the estimations obtained from our model. We keep one core on and run a computation-intensive job on the core. Then, we measure the power consumption at different frequency levels for each cluster through the on-chip power sensors. Fig. 2 plots two curves for the PE (big) cluster, one for the measured power consumption and another for the estimated power consumption,

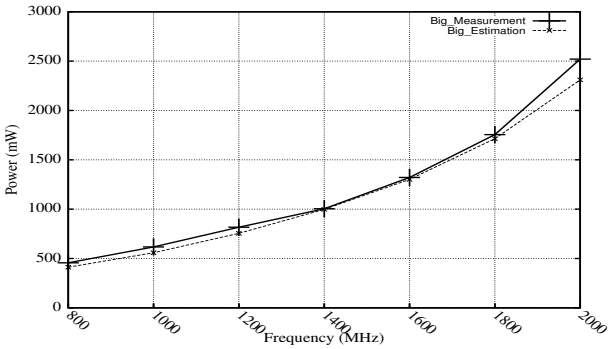

Fig. 2: Power model validation of PE (big) cluster

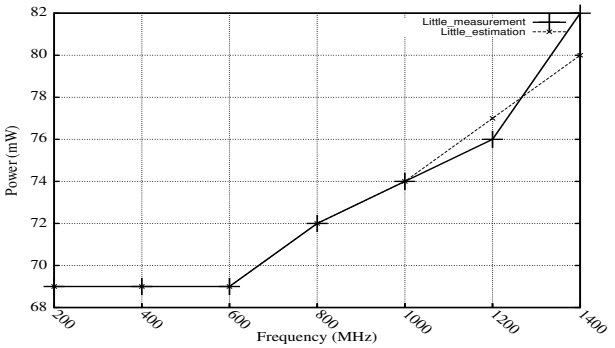

Fig. 3: Power model validation of EE (little) cluster

while Fig. 3 plots two curves for the EE (little) cluster, one for the measured power consumption and another for the estimated power consumption. In the two figures, the y-axis shows the power consumption, while the x-axis shows the different operating frequency levels. From the figures, we can see that the estimated curves are close to the measured curves, so our power model is sufficiently accurate.

To compute the total system energy consumption, we need to introduce the concept of *hyper-period* (hp),

$$hp = \text{lcm}(\breve{T}_1, \breve{T}_2, , \ldots, \breve{T}_i) \tag{6}$$

where the lcm is the *least common multiple*. For periodic tasks, every *hyper-period* has the same workload, i.e, all tasks will execute for a certain number of times. Hence, with the definition of *hyper-period*, we can build the energy model of an MPSoC within one *hyper-period* as follows:

$$E = E_s + E_d \tag{7}$$

where E_s is the total static energy consumption and the total 'uncore' energy consumption which is computed as:

$$E_s = hp\Big(\sum_{j=1}^{N_{\text{ac}}^{\text{EE}}} P_s^{\text{EE}}(f_j) + \sum_{k=1}^{N_{\text{ac}}^{\text{PE}}} P_s^{\text{PE}}(f_k) + N_{\text{ac}}^{\text{EE}} N_p^{\text{EE}} \beta^{\text{EE}} + N_{\text{ac}}^{\text{PE}} N_p^{\text{PE}} \beta^{\text{PE}}\Big) \tag{8}$$

$N_{\text{ac}}^{\text{EE}}$ is the number of active EE clusters and $N_{\text{ac}}^{\text{PE}}$ denotes the number of active PE clusters. N_p^{PE} and N_p^{EE} denote the number of processor per PE cluster and EE cluster, respectively. f_j and f_k are the operating frequency levels for the corresponding EE cluster and PE cluster, respectively. β^{EE} and β^{PE} are the power parameters shown in the last column of Table III.

The total dynamic energy consumption E_d in Eq. (7) is computed as:

$$\begin{aligned} E_d &= hp\Big(\sum_{j=1}^{N_{\text{ac}}^{\text{EE}}} \sum_{\forall \tau_i \in V_j^{\text{EE}}} \frac{C_i^{\text{EE}}}{T_i} \alpha^{\text{EE}} (f_j)^{b_{\text{EE}}} + \sum_{k=1}^{N_{\text{ac}}^{\text{PE}}} \sum_{\forall \tau_i \in V_k^{\text{PE}}} \frac{C_i^{\text{PE}}}{T_i} \alpha^{\text{PE}} (f_k)^{b_{\text{PE}}}\Big) \\ &= hp \sum_{j=1}^{N_{\text{ac}}^{\text{EE}}} U_j^{\text{EE}} \alpha^{\text{EE}} (f_j)^{b_{\text{EE}}} + hp \sum_{k=1}^{N_{\text{ac}}^{\text{PE}}} U_k^{\text{PE}} \alpha^{\text{PE}} (f_k)^{b_{\text{PE}}} \end{aligned} \tag{9}$$

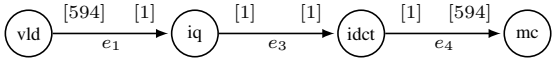

Fig. 4: H.263 Decoder

where f_j and f_k are the operating frequencies of the corresponding EE cluster and PE cluster, respectively. α^{EE}, b_{EE}, α^{PE} and b_{PE} are the estimated power parameters for an EE cluster and a PE cluster, shown in Table III.

V. Proposed Mapping Algorithm

In this section, we present our mapping algorithm, called Frequency Driven Mapping (FDM), which is able to energy-efficiently map real-time streaming applications to cluster heterogeneous MPSoCs while guaranteeing the throughput and latency constraints. The complete FDM algorithm is given in Algorithm 4. Before we explain FDM in details in Section V-D, we would like to introduce the three foundations of FDM which are described in Section V-A, V-B, V-C below.

A. Processor Type Assignment

In a heterogeneous MPSoC, choosing the type of processor for each task in an application is crucial. To support this statement, we show an example with a real-life streaming application, H.263 decoder, which is modeled as an SDF graph which is a subset of CSDF. Hence, the HRT scheduling explained in Section IV-B is applicable to the SDF. The SDF-modeled H.263 decoder consists of 4 tasks/actors and 3 edges, as shown in Fig. 4. The parameters of each actor in the H.263 decoder are shown in Table IV, where C_i^{PE} and C_i^{EE} denote the WCETs in clock cycles (cc) of the actor on the PE cluster and EE cluster, respectively. q_i is the repetition value which is used to compute the workload explained in Definition 1.

TABLE IV: The parameters of H.263

	$C_i^{\text{PE}}(cc)$	$C_i^{\text{EE}}(cc)$	q_i
vld	26018	52036	1
iq	559	1118	594
idct	500	1000	594
mc	10958	21916	1

In Table V, different processor type assignments for all actors are presented. Column 1 shows the number identifying processor type assignments, and column 2 to 5 show which type of processor each actor is assigned to where we highlight the EE processor type. The last two columns show the latency and the throughput of these processor type assignments, which are computed by using Eq. (2) and Eq. (3) and the parameters in Table IV. In these two columns, we highlight the values which satisfy the latency and throughput constraints. Intuitively, we want to have more actors running on EE processors as long as the latency and throughput constraints are met. According to the figures in Table V, we can see that an inappropriate processor type assignment significantly degrades the system performance and violates the constraints. Looking at processor type assignments 2 and 3, assigning either task *iq* or *idct* to the EE type of processor leads to a violation of the performance constraints. On the other hands, processor type assignment 5 assigns both tasks *vld* and *mc* to the EE type of processor while the performance constraints are met. Thus, determining a good processor type assignment is essential for heterogeneous MPSoCs.

From the example given above, in order to efficiently assign actors to processor types, it is important to identify those tasks which will violate the performance constraints if assigning them to the EE type of processor. By considering the characteristics of the HRT scheduling of CSDF introduced in Section IV-B, we propose an efficient way to split tasks into two categories,

TABLE V: Different processor type assignments for the H.263 decoder

	Processor Type Assignment				L ($cycles$)	$\mathcal{R}$(token/ $cycles$)
	vld	*iq*	*idct*	*mc*		
1	**EE**	PE	PE	PE	**996697**	**1/332046**
2	PE	**EE**	PE	PE	1993394	1/664092
3	PE	PE	**EE**	PE	1783000	1/594000
4	PE	PE	PE	**EE**	**996697**	**1/332046**
5	**EE**	PE	PE	**EE**	**996697**	**1/332046**

bottleneck and non-bottleneck tasks. The bottleneck actors/tasks should be assigned to PE processors in order to guarantee the performance, while the non-bottleneck actors/tasks can be assigned to EE processors for the purpose of energy saving. We introduce the following proposition:

Proposition 1. *For a CSDF graph scheduled using hard-real-time scheduling, increasing WCET C_i of task τ_i will not increase the latency and reduce the throughput, if the maximum workload $\hat{W}$ remains the same.*

Proof: By looking at Eq. (2), we can see that the latency is only determined by the start times and periods of the input and output actors. On the one hand, from Eq. (1), it is not difficult to see that $\hat{W}$ is the variable part to compute period $\breve{T}_i$, because q_i and $lcm(\overrightarrow{q})$ are both constants. Hence, as long as the maximum workload $\hat{W}$ does not increase, increasing other actors' WCETs will not change any actor's period. As a result, the throughput will not be reduced. On the other hand, start time S_i depends on the data-dependency and the deadlines of precedent actors. The data-dependency will not change in any case, while $D_i = \breve{T}_i$ and period $\breve{T}_i$ does not change. Hence, S_i remains the same as well. As a result, the latency does not increase. ■

It follows from Proposition 1 that some actors in the graph can execute slowly, while not degrading the application performance. Thus, Proposition 1 can help us to classify the actors into the two categories mentioned above. If the actor is assumed to be executed on an EE processor (longer WCET) and its new workload W_i does not change the maximum workload $\hat{W}$, then it is a non-bottleneck actor and can be assigned to an EE cluster without degrading the application performance. Otherwise, the actor should be assigned to a PE cluster in order to guarantee the performance. We look back at the example of the H.263 decoder. From Table V, since executing *vld* and *mc* on the EE type of processor does not lead to an increase of $\hat{W}$, this assignment does not violate the performance constraints. Therefore, we can use $\hat{W}^{\text{PE}}$ as a threshold to determine which type of processor an actor should be run on, where $\hat{W}^{\text{PE}}$ is the **maximum actor workload** assuming that all actors run on PE processors. Algorithm 1 presents a pseudo-code showing how to classify the actors, where V^{EE} and V^{PE} denote actors assigned to EE type and PE type of processors, respectively. To reduce the complexity of the processor type assignment, first we sort the actors in order of increasing workload assuming all of them are assigned to EE processors - see Line 1 in Algorithm 1. Then, with the sorted actors, we use $\hat{W}^{\text{PE}}$ as the threshold and deploy a binary search algorithm to find the pivotal point by which we can split the sorted actors into two sets, one for the EE type of processor and another for the PE type of processor. Since it is impossible to guarantee that the binary search can always find one actor whose W_i^{EE} is equal to $\hat{W}^{\text{PE}}$, we pick up the one with the biggest index as the pivotal point, where the condition $W_i^{\text{EE}} \leq \hat{W}^{\text{PE}}$ is met. Since the sorting algorithm has a complexity of $O(|V| \log |V|)$ and the complexity of the binary search is $O(\log |V|)$, the complexity of Algorithm 1 is $O(|V| \log |V|)$.

The processor type assignment can: (1) assign actors of an

Algorithm 1: Processor Type Assignment

input : $G = (V, E)$
output: V^{EE} and V^{PE}

$V \leftarrow$ Sort $\forall \tau_i \in V$ in increasing order of W_i^{EE} of τ_i
$b \leftarrow$ Binary search to find the position in V with the biggest index , where actor τ_i can meet $W_i^{\text{EE}} \leq \hat{W}^{\text{PE}}$.
$V^{\text{EE}} \leftarrow V[0 : b]$
$V^{\text{PE}} \leftarrow V - V^{\text{EE}}$
return V^{EE} and V^{PE}

TABLE VI: Different mappings for H.263 decoder

	PE Clusters	Actor Mapping				Energy Consumption (μJ)
		vld	*iq*	*idct*	*mc*	
WFD	2	PE1	PE2	PE1	PE1	1378
FFD	1	PE1	PE1	PE1	PE1	1226

application graph to two different types of processors and (2) allow to initially decide whether the system has enough resources to schedule this application. Suppose that U^{EE} and U^{PE} are the total utilization of V^{EE} and V^{PE} returned by Algorithm 1, respectively. If $U^{\text{EE}} > N_c^{\text{EE}} \times N_p^{\text{EE}}$, the tasks from V^{EE} are not schedulable on EE clusters. If tasks on the EE clusters are not schedulable, we can move some of them to PE clusters such that the tasks can run on the system. Based on Proposition 1, it is trivial to observe that reassigning the tasks in set V^{EE} to set V^{PE}, i.e., assigning these tasks to the PE type of cluster, is still able to guarantee the performance constraints. However, if tasks on the PE clusters are not schedulable, i.e., $U^{\text{PE}} > N_c^{\text{PE}} \times N_p^{\text{PE}}$, that means that with the throughput and latency constraints the application is not schedulable on the system.

B. Task mapping

When the processor type assignment is determined as described in Section V-A, tasks need to be mapped onto clusters. The task mapping is analogous to a bin-packing problem which is known to be an NP-hard problem [14]. Several well-known heuristic algorithms for the bin-packing problem, e.g., first-fit, best-fit, etc, have been proposed. In terms of energy efficiency, the worst-fit-decreasing (WFD) algorithm is evaluated as the best mapping heuristic [29] for the partitioned scheduling. [11] also uses an WFD-like approach. However, by using the following example, we will show that WFD does not work very well in the context of a cluster MPSoC with cluster scheduling.

Here, we use a cluster homogeneous MPSoC to illustrate this problem, where the homogeneous MPSoC has two PE clusters, each with four identical PE processors. Table VI shows two mappings for the H.263 decoder in Fig. 4 by using two different mapping algorithms, worst-fit-decreasing (WFD) and first-fit-decreasing (FFD). The mapping derived by WFD consumes more energy than the mapping obtained by FFD. The reason is that WFD tries to distribute the heavy tasks to different clusters, where these heavy tasks have large utilization and need a high operating frequency in order to meet their deadlines. In the context of a cluster MPSoC, these heavy tasks constrain the minimum operating frequency of the cluster. On the contrary, FFD always strives to find the first available cluster to map, where this strategy is more likely to map the heavy tasks with the same or close utilization to the same cluster. Then, the system can efficientlly utilize cluster DVFS to reduce the energy consumption. Hence, in our approach we use FFD to map tasks to clusters in order to obtain an initial mapping. Given this initial mapping from FFD, we can compute the minimum operating frequency of a cluster. However, the frequency of a cluster is not only determined by the task with the largest utilization, but also the total utilization of the tasks has to be taken into account. The following example shows the effect of the total utilization.

Example 1. *Consider a cluster with two processors and three tasks with utilization* $\{0.5, 0.5, 0.5\}$. *The three tasks can be mapped to the cluster because the total utilization of the tasks is* $0.5 + 0.5 + 0.5 < 2$. *The frequency of this cluster however can not be set according to the task's maximum utilization which is 0.5, because the total utilization is 1.5 and the utilization bound (the number of processor) is 2. If the frequency is scaled with 0.5, then the total utilization of this task set becomes* $\frac{1.5}{0.5} > 2$ *which means the task set is not schedulable on the cluster.*

Considering the example above, the frequency of a cluster should be computed as follows:

$$f_j = \max(\max_{\forall \tau_i \in V_j} \{u_i\}, \frac{U_j}{N_p}) \times f_{\max} \tag{10}$$

where u_i is the utilization of task $\tau_i \in V_j$ and V_j is the set of tasks mapped to cluster j. $f_{\max}$ is the maximum frequency of the type of processor used in the cluster. U_j is the total utilization of V_j and N_p is the number of processors in the cluster. Usually, a cluster only supports a set of finite discrete frequency levels. Hence, we select the minimum frequency from the frequency set which is greater than or equal to frequency f_j in Eq. (10).

With Eq. (10), we classify the clusters into two categories, namely U-cluster and T-cluster, which later will be used in our remapping phase described in Section V-C.

Definition 2. *An U-cluster is a cluster where* $\max_{\forall \tau_i \in V_j} \{u_i\} < \frac{U_j}{N_p}$.

U-cluster means that the operating frequency of the cluster is determined by the total utilization.

Definition 3. *A T-cluster is a cluster where* $\max_{\forall \tau_i \in V_j} \{u_i\} \geq \frac{U_j}{N_p}$.

T-cluster means that the operating frequency of the cluster is determined by the task which has the largest utilization. Hence, we call such task a **constrained task**.

Definition 4. *In a T-cluster, a* **constrained task** *is the task which has the largest utilization.*

C. Remapping

The FFD algorithm described in Section V-B enables to quickly map tasks to clusters. On a given MPSoC platform, FFD might just use a few clusters to run the application tasks and leave the rest of the available clusters unused. This would lead to a few used clusters with high utilization whose frequency can not be scaled down. Hence, based on the FFD mapping, we propose a remapping approach to explore the possibility to energy-efficiently utilize the unused clusters on the system such that we can balance or offload the workload of some clusters to the unused clusters in order to further scale down the clusters' operating frequencies to reduce the total system energy consumption.

Our remapping approach is based on analysis for the U-cluster and T-cluster categories introduced and defined in Section V-B. Below, we discuss how to remap tasks for both categories of clusters in order to reduce the energy consumption.

1) U-cluster: According to Definition 2, in an U-cluster, the frequency of the cluster is determined by the total utilization. Hence, we strive to remap some tasks to an unused cluster to reduce the total utilization, which in turn allows to scale down the frequency further. In order to minimize the energy consumption, we need to find how many tasks should be remapped and what the optimal frequencies are for the initial cluster and the new cluster to be used.

Consider that we have an initial U-cluster onto which a task set with utilization U is mapped. We split the task set into two

Algorithm 2: Split tasks for U cluster

input : A task set Γ
output: two tasksets Γ_1 and Γ_2
1 Sort tasks in Γ in order of decreasing utilization;
2 $\Gamma_1 \leftarrow \Gamma, \quad \Gamma_2 \leftarrow \emptyset$;
3 **for** $i = 1$ *to* $|\Gamma|$ **do**
4 **if** $U_2 \leq U_1$ **then**
5 $\Gamma_1 \leftarrow \Gamma_1 - \tau_i$;
6 $\Gamma_2 \leftarrow \Gamma_2 + \tau_i$;
7 **else**
8 Break;
9 **return** Γ_1 *and* Γ_2*;*

Algorithm 3: Split tasks for T-cluster

input : A task set Γ
output: two tasksets $\Gamma 1$ and Γ_2
1 Sort tasks in Γ in order of decreasing utilization;
2 $\Gamma_1 \leftarrow \emptyset, \quad \Gamma_2 \leftarrow \emptyset$;
3 **for** $i = 1$ *to* $|\Gamma|$ **do**
4 **if** τ_i *can run at a lower frequency* **then**
5 **for** $j = i$ *to* $|\Gamma|$ **do**
6 $\Gamma_1 \leftarrow \Gamma_1 + \tau_j$;
7 $\Gamma_2 \leftarrow \Gamma - \Gamma_1$;
8 Break;
9 **return** Γ_1 and Γ_2;

subsets, one with utilization U_1 and another with U_2. We remap the task set with U_2 to an unused cluster, and keep the task set with U_1 on the initial cluster. Then the energy consumption of the new mapping can be computed as follows:

$$E = hp\big(U_1\alpha f_1^b + N_p\beta + P_s(f_1) + U_2\alpha f_2^b + N_p\beta + P_s(f_2)\big) \tag{11}$$

where f_1 is the operating frequency of the initial cluster, and f_2 is the operating frequency of the new cluster. In Eq. (11), there are four variables, U_1, U_2, f_1, and f_2. Since frequencies f_1 and f_2 depend on U_1 and U_2, respectively, and U_1 is related to U_2, these interrelationship between them makes it difficult to find optimal values for all variables in order to minimize Eq. (11). Hence, with the consideration of simplifying the procedure, we use a load balancing approach to split tasks on an U-cluster into two tasksets. The split tasksets have close total utilizations. Algorithm 2 presents the pseudo-code of splitting the tasks for an U-cluster. We first sort tasks in order of decreasing utilization. Then, we assign the tasks one by one to the taskset Γ_2. As soon as the utilization U_2 of Γ_2 is greater than or equal to the utilization U_1 of Γ_1, the algorithm terminates and returns Γ_1 and Γ_2. Due to the sorting algorithm used in Line 1, the complexity of Algorithm 2 is $O(|\Gamma|\log(|\Gamma|))$.

The remapping will switch on a new cluster, so the remapping should provide enough energy reduction to compensate the static and 'uncore' power consumption of the new cluster. In order to test whether it is worthwhile remapping tasks to an unused cluster, we provide the following proposition to validate the efficiency of the remapping.

Proposition 2. *Given a taskset Γ and its subsets Γ_1 and Γ_2, their utilizations are U, U_1, and U_2, respectively, where $U_1 + U_2 = U$, and taskset Γ is assigned to one cluster. The cluster is an U-cluster. Then, moving taskset Γ_2 to an unused cluster can reduce the energy consumption, if the following condition is met:*

$$U_1\alpha(f^b - f_1^b) + U_2\alpha(f^b - f_2^b) > P_s(f_1) + P_s(f_2) + N_p\beta - P_s(f) \tag{12}$$

where f is the operating frequency of the initial cluster before remapping, and f_1 and f_2 are the operating frequencies of the initial cluster and the new cluster after remapping, respectively.

Proof: Before the remapping, the energy consumption of the initial cluster is as follows:

$$E = hp\big(U\alpha f^b + N_p\beta + P_s(f)\big) \tag{13}$$

After the remapping, the energy consumption of the two clusters is:

$$E_n = hp\big(U_1\alpha f_1^b + N_p\beta + P_s(f_1) + U_2\alpha f_2^b + N_p\beta + P_s(f_2)\big) \tag{14}$$

To guarantee that the remapping leads to less energy consumption, we need the following inequality satisfied:

$$E > E_n$$

Since we consider symmetric clusters, by substituting Eq. (13) and (14) in the inequality and eliminating the same terms on both sides, we obtain the following condition:

$$U_1\alpha(f^b - f_1^b) + U_2\alpha(f^b - f_2^b) > P_s(f_1) + P_s(f_2) + N_p\beta - P_s(f) \tag{15}$$

■

2) T-cluster: According to Definition 3, in a T-cluster, the frequency of the cluster is determined by the **constrained task** (Definition 4). However, within one cluster, the FFD discussed in Section V-B might map some other tasks which have lower utilization and could operate at a lower frequency. In this case, remapping these tasks to an unused cluster operated at a lower frequency may result in an overall reduced energy consumption.

Example 2. *Given two clusters with two processors each and a task set with three tasks where the utilizations are $\{0.9, 0.3, 0.3\}$, the FFD maps all tasks to one cluster, and the cluster is a T-cluster. The frequency is determined by the task with utilization 0.9. However, if we map the two tasks with utilization 0.3 to the unused cluster, the frequency of the initial cluster is not changed, but the new cluster operates at a lower frequency which can significantly reduce the overall energy consumption.*

Example 2 illustrates how we can remap tasks of a T-cluster. We find the actors which can run at a frequency lower than the current cluster frequency and remap them to an unused cluster. Algorithm 3 presents the pseudo-code of splitting tasks of a T-cluster. The complexity of Algorithm 3 is $O(|\Gamma|\log|\Gamma|)$ due to the sorting algorithm used in Line 1. The remapping will need more static power consumption and 'uncore' power consumption due to the new cluster switched on. Thus, the efficiency of the remapping should be verified. The following proposition presents an efficient way to check this.

Proposition 3. *Given a taskset Γ and its subsets Γ_1 and Γ_2, their utilizations are U, U_1, and U_2 respectively, where $U = U_1 + U_2$, and taskset Γ is assigned to one cluster. The cluster is a T-cluster and the constrained actor is in subset Γ_2. Then, moving taskset Γ_1 to an unused cluster can reduce the energy consumption, if the following condition is met:*

$$U_1 \cdot \alpha \cdot f^b > U_1 \cdot \alpha \cdot f_1^b + N_p\beta + P_s(f_1) \tag{16}$$

where f and f_1 are the operating frequencies of the initial and new cluster, respectively.

Proof: Assume that taskset Γ is assigned to one cluster and its operating frequency is f. Before the remapping, the energy consumption of the initial cluster can be computed as follows:

$$E = hp\big(U \cdot \alpha \cdot f^b + N_p\beta + P_s(f)\big) \tag{17}$$

If taskset Γ_1 which is a subset of Γ is remapped to an unused cluster, the operating frequency of the new cluster is f_1. Since the constrained task is in taskset Γ_2 and taskset Γ_2 remains on the initial cluster, the frequency of the initial cluster does not change. After the remapping, the energy consumption of the two

clusters is the following:

$$E_n = hp\big(U_2\cdot\alpha\cdot f^b + N_p\beta + P_s(f) + U_1\cdot\alpha\cdot f_1^b + N_p\beta + P_s(f_1)\big) \quad (18)$$

The assignment with two clusters is more energy-efficient, if $E > E_n$. Since $U = U_1 + U_2$, we replace U_2 with $U - U_1$ in Eq. (18). By substituting Eq. (17) and (18) and eliminating the same terms on both sides of inequality $E > E_n$, we obtain:

$$U_1 \cdot \alpha \cdot f^b > U_1 \cdot \alpha \cdot f_1^b + N_p\beta + P_s(f_1) \quad (19)$$

■

D. The FDM Algorithm

In this section, we present our overall mapping algorithm, called Frequency Driven Mapping (FDM). The inputs to FDM are a CSDF graph and a cluster heterogeneous MPSoC, and the outputs are the task mapping to clusters and the minimum operating frequency for each cluster which is active. Algorithm 4 shows the pseudo-code of FDM. In Line 1, FDM applies Algorithm 1 explained in Section V-A to split tasks into two sets V^{PE} and V^{EE} which denote the tasks assigned to PE type and EE type of processors, respectively. In Algorithm 1, the processor type assignment completes the assignment procedure with the guarantees of the performance constraints. Hence, if the task sets V^{EE} and V^{PE} derived by Algorithm 1 are schedulable on the given MPSoC, the throughput and latency constraints are met for the application. From Line 2 to 5, we check whether the input MPSoC has enough resources to schedule the real-time streaming application. If there is no enough EE type of processors, we select some tasks from set V^{EE} and assign them to set V^{PE} such that we have enough EE processors to schedule the tasks in set V^{EE}. The tasks are selected in order of decreasing utilization, and the selection is terminated as soon as the tasks in set V^{EE} are schedulable on the EE processors. However, if there is no enough PE type of processors, that means the application is not schedulable on the input MPSoC. The algorithm terminates and signals failure at Line 5. After this schedulability check, we use the FFD heuristic discussed in Section V-B on PE clusters and EE clusters to map tasks to clusters in Line 6. After this phase, we obtain the initial mapping and the corresponding active cluster set, i.e., Φ_{ac} shown in Line 6. An element in set Φ_{ac} is a cluster which includes the tasks mapped to the cluster and the operating frequency of the cluster computed by Eq. (10).

With the obtained initial mapping, a remapping procedure starts from Line 7. In this procedure, we go through every cluster in the active cluster set Φ_{ac} to check what category the cluster falls into, U-cluster or T-cluster. From Line 8 to 14, we do remapping for an U-cluster. At Line 8 and 9, if a cluster is an U-cluster of type EE or PE and there is an unused cluster of the same type available (PE or EE), we split the tasks into two sets by using Algorithm 2 and we use Proposition 2 to validate the remapping in Line 10. If the remapping leads to energy reduction, we complete the remapping. Otherwise, the mapping remains unchanged. From Line 15 to 21, we do remapping for a T-cluster. For a T-cluster, we use Algorithm 3 to split tasks in Line 16 and we use Proposition 3 to validate the remapping in Line 17. Note that after we do a remapping the new cluster ϕ_{unused} is added to the active cluster set Φ_{ac} shown in Line 12 and 19. Then later the new cluster also will undertake the remapping procedure as long as there is an unused cluster of the same type and it can meet the remapping conditions. Finally, FDM updates the operating frequencies of each cluster in set Φ_{ac} in Line 22 by using Eq. (10). At Line 23, FDM outputs the final mapping and the operating frequency of each active cluster. Since the complexity of Algorithm 2 and 3 are both $O(|V|\log(|V|))$, in the worst case the complexity of FDM is $O(N \times |V|\log(|V|))$, where N is the total number of clusters in the input MPSoC and $|V|$ is the total number of actors in the input CSDF graph.

Algorithm 4: Frequency Driven Mapping

input : A CSDF graph G and a cluster heterogeneous MPSoC
output: A task mapping for each cluster and the minimum operating frequency for each active cluster

1 $V^{\text{PE}}, V^{\text{EE}} \leftarrow$Apply Algorithm 1 to split all actors $\tau_i \in V$ to two parts;
2 **if** $\lceil U^{EE} \rceil > N_c^{EE} \times N_p^{EE}$ **then**
3 Map some actors τ_i to PE clusters in order of decreasing utilization such that $\lceil U^{\text{EE}} \rceil \leq N_c^{\text{EE}} \times N_p^{\text{EE}}$;
4 **if** $\lceil U^{PE} \rceil > N_c^{PE} \times N_p^{PE}$ **then**
5 **return** Unschedulable;
6 $\Phi_{ac} \leftarrow$Apply FFD on PE clusters and EE clusters to generate an initial task mapping and compute the frequency of each active cluster by using Eq. (10);
/* Remapping procedure */
7 **for** *j=1 to* $|\phi_{ac}|$ **do**
8 **if** $\max_{\forall \tau_i \in V_j}\{u_i\} < \frac{U_j}{N_p}$ *& an unused cluster of the same type is available* **then**
// U-cluster
9 $V_{j,1}, V_{j,2} \leftarrow$ Apply Algorithm 2 to split tasks;
10 **if** $V_{j,1}$ *and* $V_{j,2}$ *can meet the condition in Proposition 2* **then**
11 Keep $\forall \tau_i \in V_{j,1}$ on the initial cluster and remap $\forall \tau_i \in V_{j,2}$ to unused cluster ϕ_{unused};
12 $\Phi_{ac} \leftarrow \Phi_{ac} + \phi_{\text{unused}}$;
13 **else**
14 Keep the initial mapping;
15 **if** $\max_{\forall \tau_i \in V_j}\{u_i\} \geq \frac{U_j}{N_p}$ *& an unused cluster of the same type is available* **then**
// T-cluster
16 $V_{j,1}, V_{j,2} \leftarrow$ Apply Algorithm 3 to split tasks;
17 **if** $V_{j,1}$ *can meet the condition in Proposition 3* **then**
18 Keep $\forall \tau_i \in V_{j,2}$ on the initial cluster and remap $\forall \tau_i \in V_{j,1}$ to unused cluster ϕ_{unused};
19 $\Phi_{ac} \leftarrow \Phi_{ac} + \phi_{\text{unused}}$;
20 **else**
21 Keep the initial mapping;
22 Update the operating frequencies of clusters in set Φ_{ac} by using Eq. (10);
23 **return** Φ_{ac};

TABLE VII: The Streaming Applications

APP	$\lvert V\rvert$	$\lvert E\rvert$	L $(cycles)$	$\mathcal{R}$(token/ $cycles$)
Beamformer	57	70	61152	1/5076
Bitonicsort	40	46	2280	1/95
CHVocoder	55	70	28400	1/35550
DCT	8	7	380928	1/47616
DES	53	60	46080	1/1024
FFT	17	16	204544	1/12032
FMRadio	43	53	17208	1/1434
MP3	14	18	16795242	1/1866138
MPEG	23	26	138240	1/7680
Serpent	120	128	370296	1/3336
TDE	29	28	1071840	1/36960
Vocoder	114	147	291360	1/9105

VI. Evaluation

In this section, we present three experiments to demonstrate the efficiency of the proposed FDM algorithm compared to the existing approaches proposed in [10] and [11]. We apply our FDM and the mapping approaches from [10] and [11] on cluster heterogeneous and homogeneous MPSoCs. We choose [10] and [11] to compare with, because the mapping approaches in [10] and [11] are specifically devised for cluster MPSoCs as we consider in this work. Therefore, their work is the most related and relevant to our approach.

We select 11 real-life streaming applications from the

TABLE VIII: Cluster Heterogeneous MPSoC configurations

Configuration	Granularity	PE clusters	EE clusters
MPSoC_2_20_28	2 procs	20	28
MPSoC_4_10_14	4 procs	10	14
MPSoC_8_5_7	8 procs	5	7

StreamIt [12] benchmark suite and the MP3 decoder [30], where all streaming applications are modeled as CSDF graphs. We use the same parameters, i.e., WCETs of application tasks, as specified in [23]. An overview of all streaming applications is given in Table VII. $|V|$ denotes the number of tasks/actors in a CSDF graph, while $|E|$ denotes the number of edges. L is the minimum achievable latency and $\mathcal{R}$ is the maximum achievable throughput which are computed by using Eq. (2) and Eq. (3), when the applications are scheduled by the HRT scheduling described in Section IV-B. In our experiments, for each application, we set as constraints the corresponding minimum achievable latency and the maximum achievable throughput (L and $\mathcal{R}$ given in Table VII) and when we map the applications to the target platforms.

As target platforms, we consider three heterogeneous MPSoCs with different number of clusters and cluster granularities. We use 'MPSoC_x_pe_ee' to denote a cluster heterogeneous MPSoC, where 'x' denotes the number of processors per cluster, 'pe' and 'ee' denote the number of PE clusters and EE clusters, respectively. The three considered MPSoCs are described in Table VIII. Column 'granularity' shows the number of processors per cluster, while column 'PE clusters' and 'EE clusters' show the number of PE clusters and EE clusters in the MPSoC, respectively.

For a cluster, we use the power model described in Eq. (5), where the power parameters are given in Table III and Table II. In our experiments, we use our FDM approach and the reference mapping approaches described in [10] and [11] to map the tasks of the streaming applications to the three MPSoCs and we compute the energy consumption of each application to MPSoC mapping configuration using Eq. (7), (8) and (9). The metric for the evaluation of each configuration is the energy reduction achieved by our proposed FDM approach over the different reference mapping approaches. We use the following equation to compute the energy reduction:

$$r = \frac{E_{\text{ref}} - E_{\text{FDM}}}{E_{\text{ref}}} \tag{20}$$

where E_{ref} is the energy consumption of an application to MPSoC mapping configuration obtained by a reference mapping approach and E_{FDM} denotes the energy consumption achieved by our proposed FDM with cluster DVFS.

A. Comparison with [10] on Heterogeneous MPSoCs

In this section, we compare our proposed FDM approach to the mapping approach proposed in [10]. In [10], the authors proposed several mapping approaches for cluster heterogeneous MPSoCs, and in our experiments we select the best mapping approach evaluated in [10] and refer to it as CKR. In this experiment, the CKR is considered as the reference point and the energy reduction for each application benchmark is computed by using Eq. (20).

Fig. 5 depicts the energy reduction for each benchmark mapped on the three different MPSoCs, where the x-axis shows the benchmarks and the y-axis shows the energy reduction. For 7 out of 12 benchmarks, our proposed FDM+DVFS approach finds a mapping that consumes less energy compared to the one obtained by the CRK approach. The main reason is that the CKR approach is similar to the FFD algorithm but it does not consider remapping to efficiently utilize the unused clusters. Hence, for the 7 benchmarks, the remapping approach in our FDM algorithm outperforms the CKR. For the other 5 benchmarks, the remapping is not beneficial for them, so our proposed FDM approach achieves the same results as the CKR approach.

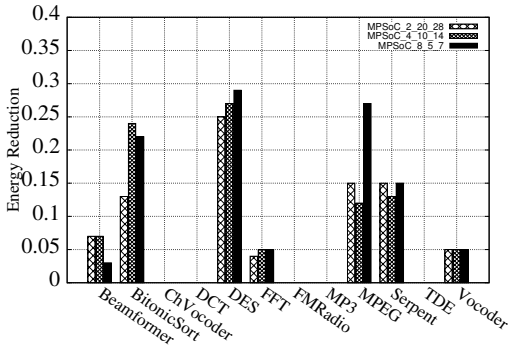

Fig. 5: Comparison between FDM+DVFS and CKR+DVFS

TABLE IX: Summary of Fig. 5

	2 procs	4 procs	8 procs
FDM+DVFS average	7%	7.7%	8.9%
Max energy reduction	25%	27%	29%

TABLE X: Summary of Fig. 6

	2 procs	4 procs	8 procs
FDM+DVFS average	6.3%	8.5%	9.4%
Max energy reduction	19%	21%	34%

The energy reduction results are summarized in Table IX. We see that the average energy reduction is 7%, 7.7%, and 8.9% for the three MPSoCs with 2, 4 and 8 processors per cluster, respectively. Among all experiments, the maximum energy reduction occurs to benchmark DES which is 25%, 27%, and 29% for the three MPSoCs with 2, 4, and 8 processors per cluster, respectively.

B. Comparison with [11] on Heterogeneous MPSoCs

In this experiment, we compare our FDM approach with the approach proposed in [11] which we refer to as KYD. Since [11] only considers cluster homogeneous MPSoCs, we apply our processor type assignment proposed in Section V-A, i.e., Algorithm 1, to determine the processor type for each actor and then utilize the KYD approach to map the actors to clusters. Thus, in this experiment, 'Algorithm 1+KYD+DVFS' is used as the reference mapping approach in Eq. (20).

The energy reduction for the different benchmarks mapped on the different MPSoCs is depicted in Fig. 6. For 7 out of 12 benchmarks, our FDM+DVFS finds a mapping which consumes less energy than the mapping approach 'Algorithm 1+KYD+DVFS'. For the rest of the benchmarks, our proposed approach finds a mapping that consumes the same energy as the reference mapping approach. For benchmarks ChannelVocoder, DCT, MP3 and FMRadio, their actors which are assigned to PE type of clusters have very similar workload, hence evenly distributing heavy tasks by the KYD approach can find the energy efficient mapping as our FDM approach does. For benchmark Vocoder, only two heavy tasks are assigned to PE clusters and running them on different clusters leads to energy efficiency, so the KYD approach can find the best mapping by mapping these two tasks to two different clusters as our FDM approach does.

The results are summarized in Table X. We can see that the average energy reduction of the three MPSoCs is 6.3% for the MPSoC with 2 processors per cluster, 8.5% for the MPSoC with 4 processors per cluster, and 9.4% for the MPSoC with 8 processors per clusters. The maximum energy reduction is 19% in benchmark Beamformer for 2 processors per cluster, 21% and 34% in benchmark DES for 4 processors per cluster and 8 processors per cluster, respectively.

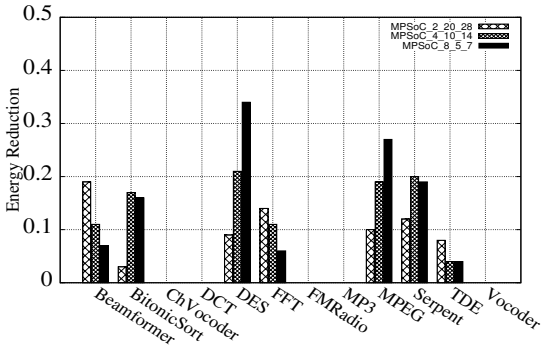

Fig. 6: FDM+DVFS vs. Algorithm 1+KYD+DVFS

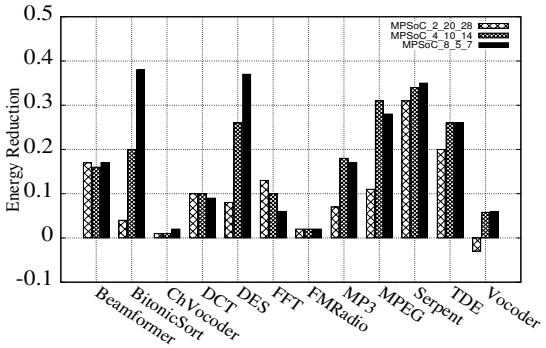

Fig. 7: FDM+DVFS vs. KYD+DVFS on homo MPSoCs

C. Comparison with [11] on Homogeneous MPSoCs

The KYD approach [11] is originally proposed for cluster homogeneous MPSoCs. In order to have a fair comparison, we apply our FDM approach to homogeneous MPSoCs and compare it with the KYD approach to show the efficiency of our FDM approach. Since we need to guarantee the throughput and latency constraints shown in Table VII, running the benchmarks on a cluster homogeneous MPSoC comprised of EE clusters will violate the performance constraints. Thus, we only map the applications to the PE clusters available in the cluster MPSoCs described in Table VIII, thereby considering cluster homogeneous MPSoCs that can meet the performance constraints for every application.

Fig. 7 depicts the energy reduction of each benchmark mapped on the three different MPSoCs by only using the PE clusters. In Fig. 7, we see that for benchmark Vocoder on platform 'MPSoC_2_20_28', the mapping derived by our FDM approach consumes 3% more energy than the KYD approach. With this fine granularity of the cluster size, i.e, the small number of processors per cluster, benchmark Vocoder has a lot of mapping possibilities. Since the KYD has a design space exploration which enables to explore more mappings and our FDM only improves the mapping generated by the FFD heuristic, in the case of benchmark Vocoder our FDM does not find a more energy efficient mapping compared to KYD. However, except benchmark Vocoder on platform 'MPSoC_2_20_28', our FDM approach outperforms the KYD in all other cases by finding more energy efficient mappings.

The results of this experiment are summarized in Table XI. We see that for different MPSoCs our FDM approach can reduce the energy consumption by an average of 10%, 16.6% and 18.5%. The maximum reduction occurs for benchmark Serpent which is 31% and 34% for the MPSoCs with 2 and 4 processors per cluster, respectively. For the MPSoC with 8 processors per cluster, benchmark Bitonicsort has the maximum energy reduction which is 38%.

TABLE XI: Summary of Fig. 7

	2 procs	4 procs	8 procs
FDM+DVFS average	10%	16.6%	18.5%
Max energy reduction	31%	34%	38%

VII. Conclusions

In this paper, we proposed a polynomial time algorithm to energy-efficiently map real-time streaming applications with latency and throughput constraints to cluster heterogeneous MPSoCs. Compared with existing approaches on cluster heterogeneous MPSoCs and cluster homogeneous MPSoCs, the experimental results show that our proposed approach outperforms the existing approaches by finding a more energy efficient mapping. Our approach can save up to 34% and 38% more energy on the cluster heterogeneous MPSoCs and the cluster homogeneous MPSoCs, respectively.

References

[1] G. Kahn, "The semantics of a simple language for parallel programming," in *Proc. of Information Processing*. North-Holland, 1974.
[2] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.
[3] G. Bilsen *et al.*, "Cyclo-static dataflow," *IEEE Trans. Signal Process.*, vol. 44, no. 2, pp. 397–408, 1996.
[4] A. K. Singh *et al.*, "Mapping on multi/many-core systems: survey of current and emerging trends," in *DAC*, 2013.
[5] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ISLPED*, Aug 2007.
[6] ARM, "http://www.arm.com."
[7] J. Howard *et al.*, "A 48-core ia-32 message-passing processor with dvfs in 45nm cmos," in *ISSCC*, Feb 2010, pp. 108–109.
[8] R. Kumar *et al.*, "Single-isa heterogeneous multi-core architectures: the potential for processor power reduction," in *MICRO*, 2003.
[9] Exynos 5422, "http://www.samsung.com/."
[10] A. Colin *et al.*, "Energy-efficient allocation of real-time applications onto heterogeneous processors," in *RTCSA*, 2014.
[11] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *DATE*, March 2011, pp. 1–6.
[12] W. Thies *et al.*, "An empirical characterization of stream programs and its implications for language and compiler design," in *PACT*, 2010.
[13] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, p. 35, 2011.
[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.
[15] J. Anderson and A. Srinivasan, "Pfair scheduling: beyond periodic task systems," in *RTCSA*, 2000.
[16] H. Cho *et al.*, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS*, 2006.
[17] A. Bastoni *et al.*, "An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers," in *RTSS*, 2010.
[18] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in *RTCSA*, 2007.
[19] A. K. Singh *et al.*, "Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems," in *DAC*, 2013.
[20] R. Xu *et al.*, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *RTSS*, 2007, pp. 25–38.
[21] Y. Wang *et al.*, "Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip," *TODAES*, 2011.
[22] G. Chen *et al.*, "Energy optimization with worst-case deadline guarantee for pipelined multiprocessor systems," in *DATE*, 2013.
[23] M. Bamakhrama and T. Stefanov, "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications," in *EMSOFT*, 2011.
[24] M. Bamakhrama *et al.*, "A methodology for automated design of hard-real-time embedded streaming systems," in *DATE*, 2012, pp. 941–946.
[25] J. T. Zhai *et al.*, "Exploiting just-enough parallelism when mapping streaming applications in hard real-time systems," in *DAC*, 2013.
[26] S. K. Baruah *et al.*, "Proportionate progress: A notion of fairness in resource allocation," in *STOC*, 1993.
[27] ODROID, "http://www.hardkernel.com/."
[28] V. Gupta *et al.*, "The forgotten 'uncore': On the energy-efficiency of heterogeneous cores," in *USENIX ATC*, 2012.
[29] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *IPDPS*, 2003, p. 113.
[30] S. Stuijk *et al.*, "SDF^3: SDF For Free," in *ACSD*, June 2006, pp. 276–278.