Latency, energy, and schedulability of real-time embedded systems

Liu, D.; Liu D.

Chapter 4

Energy Optimization for Real-Time Streaming Applications

Di Liu, Jelena Spasic, Gang Chen and Todor Stefanov,
"Energy-efficient mapping of real-time streaming applications on cluster heterogeneous MPSoCs,"
The 13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia),
Amsterdam, 2015, pp. 1-10.

With the increasing number of cores/processors fabricated on a chip, the stringent area requirement hinders to implement the core's frequency and voltage regulator hardware by using the fine-grained per-core DVFS due to its high hardware cost [HM07]. In order to balance the energy saving and hardware cost, cluster heterogeneous MPSoCs provide a promising solution. On cluster heterogeneous MPSoCs, all processors on the same cluster operate at the same frequency and voltage level, i.e., processors on the same cluster are managed by a single frequency and voltage regulator. Although the cluster heterogeneous MPSoC systems have shown their energy efficiency in the state-of-the-art chips and commercial products, e.g., Samsung Exynos 5422 [Sam16], Samsung Galaxy S7, Iphone 7 and 7Plus, etc., 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 chapter, we present our new algorithm to map streaming applications onto a cluster heterogeneous MPSoC, where the streaming applications are modeled as CSDF graphs and scheduled using the HRT scheduling framework in Section 2.3. The proposed novel algorithm is called Frequency Driven Mapping (FDM) and its main novelty is twofold:

1. By using the HRT scheduling framework for CSDF graphs, explained in Section 2.3, we propose an efficient way to determine a suitable processor type for each

actor/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 the first-fit-decreasing (FFD) heuristic and the properties of cluster MPSoCs, we remap some tasks to unused clusters in order to further reduce the energy consumption by using VFS.

We have performed various experiments on real-life streaming applications. The experimental results show that compared to the existing approaches in [CKR14] and [KYD11], the proposed FDM algorithm can achieve more energy saving.

4.1 Related Work

The energy-efficient design issue has been addressed extensively in the past decade. Voltage/frequency scaling (VFS) 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 VFS to schedule real-time tasks in an energy-efficient way. [CK07] surveys most of the papers dealing with energy efficient scheduling of real-time tasks by using VFS. All papers reported in [CK07] assume per-core VFS platforms. Regardless of the properties of cluster MPSoCs, directly adopting these per-core VFS approaches onto cluster MPSoCs may lead to an energy inefficient design/mapping [KYD11]. In contrast, the algorithm proposed in this chapter 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 [CKR14] and [KYD11] 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 4.1. First, [CKR14] and [KYD11] 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 heterogeneous MPSoCs, we consider a cluster scheduling which provides a good trade-off between global scheduling and partitioned scheduling with respect to schedulability and scalability [BBA10]. Thus, the cluster scheduling can achieve a better system utilization than partitioned scheduling. Finally, the work in [KYD11] only considers homogeneous MPSoCs, whereas we consider heterogeneous MPSoCs which provide better energy efficiency.

	heterogeneous	performance	scheduling
Colin *et al.* [CKR14]	Yes	No	Partitioned
Kong *et al.* [KYD11]	No	No	Partitioned
Our FDM approach	Yes	Yes	Cluster

Table 4.1: The difference from [KYD11] and [CKR14]

Some works have been done to reduce the energy consumption of streaming applications on MPSoCs [SDK13], [XMM07], [WLL+11] and [CHBK13], where some performance metrics are taken into account. The works in [SDK13], [XMM07], [WLL+11] and [CHBK13] use per-core VFS on homogeneous MPSoCs to reduce the energy consumption. In contrast, we consider cluster heterogeneous MPSoCs in our work. Heterogeneous MPSoCs are known to be more energy efficient than homogeneous MPSoCs [KFJ+03]. 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 VFS approaches without considering the properties of cluster MPSoCs may result in an energy inefficient mapping [KYD11].

4.2 Background

In this section, we elaborate the system model and present the power/energy model considered throughout this chapter.

4.2.1 System Model

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 [ARM16]. Each cluster has a number of identical processors, PE processor or EE processor. In total, the cluster heterogeneous MPSoC consists of $N_c^{\text{PE}} \times M_p^{\text{PE}}$ PE processors and $M_c^{\text{EE}} \times M_p^{\text{EE}}$ EE processors, where N_c^{PE} and M_p^{PE} represent the total number of PE clusters and the number of processors per PE cluster, respectively. M_c^{EE} and M_p^{EE} are the total number of EE clusters and the number of processors per EE cluster, respectively. Figure 4.1 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 chapter, we only consider symmetric clusters, i.e., all clusters have the same number of processors, but our

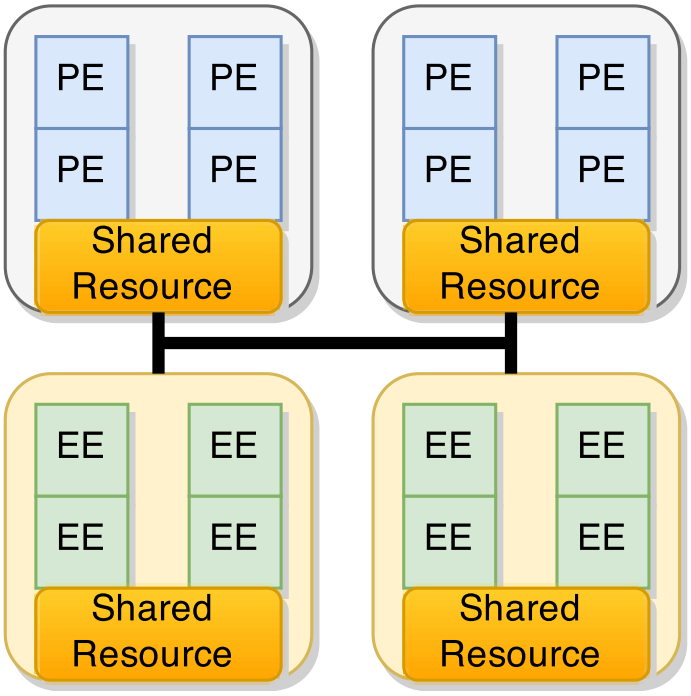

Figure 4.1: An example of a cluster heterogeneous MPSoC

approach can be easily adapted to asymmetric clusters. Since application tasks/actors may run on different processor types, the WCET C_i of a task/actor varies according to the performance of the assigned processor type. Hence, we first extend the task model described in Section 2.3 to support this WCET variation due to the heterogeneity of the systems. 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. A real-time task τ_i then is specified by a tuple of $\tau_i = \{S_i, D_i, T_i, C_i^{\text{EE}}, C_i^{\text{PE}}\}$.

As we mentioned in Section 4.1, we adopt cluster scheduling in this chapter. 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 [AS00] or LLREF [CRJ06], is deployed to schedule actors. A periodic taskset is schedulable on a homogeneous multiprocessor system by an optimal global scheduling algorithm, if $U = \sum_{\forall \tau_i \in \Gamma} C_i/T_i \leq M$ [BCPV93a], 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 $U_k^{PE} \leq M_p^{\text{PE}}$, while tasks mapped onto an EE cluster are schedulable if $U_j^{EE} \leq M_p^{\text{EE}}$. 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:

$$U_j^{\text{EE}} = \sum_{\forall \tau_i \in A_j^{\text{EE}}} \frac{C_i^{\text{EE}}}{\breve{T}_i}, \qquad U_k^{\text{PE}} = \sum_{\forall \tau_i \in A_k^{\text{PE}}} \frac{C_i^{\text{PE}}}{\breve{T}_i} \tag{4.1}$$

where A_j^{EE} and A_k^{PE} represent the set of actors assigned to EE cluster j and PE cluster k, respectively.

4.2.2 Energy Model

In this chapter, we use real measurements from the ODROID XU-3 [ODR16] board to build our power and energy models. The ODROID XU-3 has an Exynos 5422 chip [Sam16], 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' [GBK+12]. 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 4.2, where the 'uncore' power consumption for the PE (big) cluster 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 4.2, 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 4.2 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 4.2, 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,

$$P(f) = \alpha f^b + M_p \beta + P_s(f) \tag{4.2}$$

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

Table 4.2: The 'uncore' power consumption

where the first term is the dynamic power consumption, β is the static power consumption of one processor and M_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 chapter, 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 4.3.

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

Table 4.3: The estimated parameters

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. Figure 4.2 plots two curves for the PE (big) cluster, one for the measured power consumption and another for the estimated power consumption, while Figure 4.3 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 a *hyper-period* (hp),

$$hp = \text{lcm}(\breve{T}_1, \breve{T}_2, , \ldots, \breve{T}_i) \tag{4.3}$$

where the lcm is the *least common multiple* and $\breve{T}_1, \breve{T}_2, \ldots, \breve{T}_i$ are the minimum periods of application actors/tasks computed using Equation (2.8). For periodic tasks,

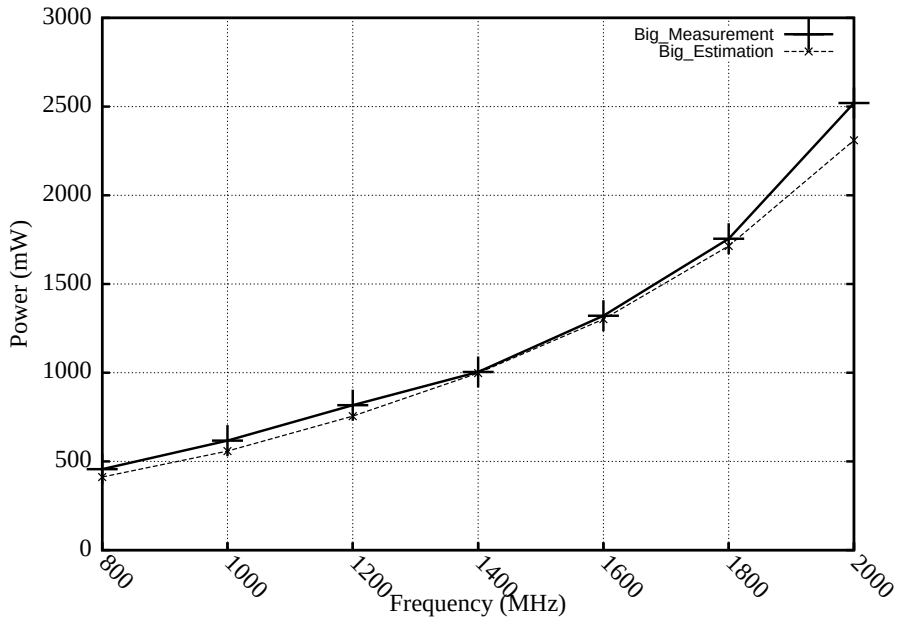

Figure 4.2: Power model validation of PE (big) cluster

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

$$E = E_s + E_d \tag{4.4}$$

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

$$E_s = hp\Big(\sum_{j=1}^{M_{\text{ac}}^{\text{EE}}} {P_s}^{\text{EE}}(f_j) + \sum_{k=1}^{M_{\text{ac}}^{\text{PE}}} P_s^{\text{PE}}(f_k) + M_{\text{ac}}^{\text{EE}} M_p^{\text{EE}} \beta^{\text{EE}} + M_{\text{ac}}^{\text{PE}} M_p^{\text{PE}} \beta^{\text{PE}}\Big) \tag{4.5}$$

$M_{\text{ac}}^{\text{EE}}$ is the number of active EE clusters and $M_{\text{ac}}^{\text{PE}}$ denotes the number of active PE clusters. M_p^{PE} and M_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 4.3.

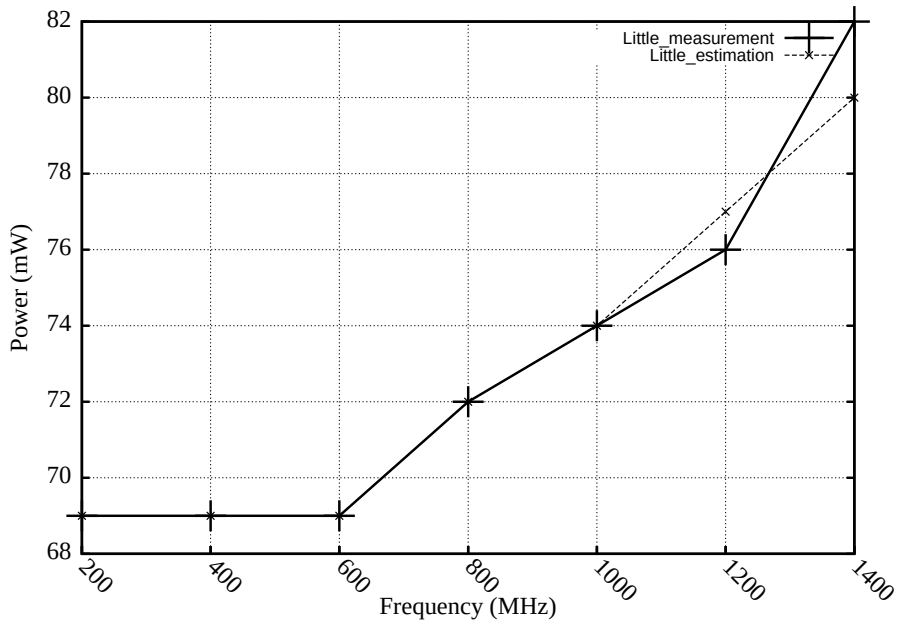

Figure 4.3: Power model validation of EE (LITTLE) cluster

The total dynamic energy consumption E_d in Equation (4.4) is computed as:

$$\begin{aligned} E_d &= hp\Big(\sum_{j=1}^{M_{\text{ac}}^{\text{EE}}} \sum_{\forall \tau_i \in A_j^{\text{EE}}} \frac{C_i^{\text{EE}}}{T_i} \alpha^{\text{EE}} f_j^{b_{\text{EE}}} \frac{f_j^{\max}}{f_j} + \sum_{k=1}^{M_{\text{ac}}^{\text{PE}}} \sum_{\forall \tau_i \in A_k^{\text{PE}}} \frac{C_i^{\text{PE}}}{T_i} \alpha^{\text{PE}} f_k^{b_{\text{PE}}} \frac{f_k^{\max}}{f_k}\Big) \\ &= hp \sum_{j=1}^{M_{\text{ac}}^{\text{EE}}} U_j^{\text{EE}} \alpha^{\text{EE}} f_j^{(b_{\text{EE}}-1)} f_j^{\max} + hp \sum_{k=1}^{M_{\text{ac}}^{\text{PE}}} U_k^{\text{PE}} \alpha^{\text{PE}} f_k^{(b_{\text{PE}}-1)} f_k^{\max} \end{aligned} \tag{4.6}$$

where f_j and f_k are the operating frequencies of the corresponding EE cluster and PE cluster, respectively. Correspondingly, $f_j^{\max}$ and $f_k^{\max}$ are the maximum operating frequencies of the EE 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 4.3.

4.3 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, modeled as CSDF graphs, to cluster heterogeneous MPSoCs while guaranteeing the

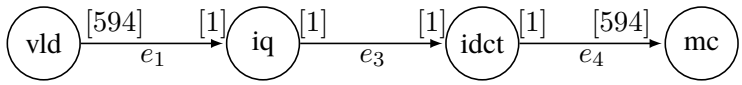

Figure 4.4: H.263 Decoder

throughput and latency constraints. The complete FDM algorithm is given in Algorithm 5 in Section 4.3.4. Before we explain FDM in details, we would like to introduce the three foundations of FDM which are described in Section 4.3.1, 4.3.2, 4.3.36.

4.3.1 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 2.3 is applicable to the SDF. The SDF-modeled H.263 decoder consists of 4 tasks/actors and 3 edges, as shown in Figure 4.4. The parameters of each actor in the H.263 decoder are shown in Table 4.4, 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 2.3.1.

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

Table 4.4: The parameters of H.263

In Table 4.5, 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 Equation (2.11) and Equation (2.12) and the parameters in Table 4.4. 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 4.5, we can see that an inappropriate processor type assignment significantly degrades the system performance and violates

	Processor Type Assignment				L	$\mathcal{R}$(token/
	vld	*iq*	*idct*	*mc*	$(cycles)$	$cycles)$
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**

Table 4.5: Different processor type assignments for the H.263 decoder

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, we propose an efficient way to split tasks into two categories, 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 Equation (2.11), 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 Equation (2.8), it is not difficult to see that $\hat{W}$ is the variable part to compute period $\check{T}_i$, 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 $D_i = \check{T}_i$ and period $\check{T}_i$ 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

Algorithm 2: Processor Type Assignment

Input: $G = (A, \mathcal{E})$
Output: A^{EE} and A^{PE}
1 $A \leftarrow$ Sort $\forall \tau_i \in A$ in increasing order of W_i^{EE} of τ_i
2 $b \leftarrow$ Binary search to find the position in A with the biggest index , where actor τ_i can meet $W_i^{\text{EE}} \leq \hat{W}^{\text{PE}}$.
3 $A^{\text{EE}} \leftarrow A[0 : b]$
4 $A^{\text{PE}} \leftarrow A - A^{\text{EE}}$
5 **return** A^{EE} and A^{PE}

classify the actors into the two categories mentioned above. If an 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 4.5, 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 2 presents a pseudo-code showing how to classify the actors, where A^{EE} and A^{PE} denote the set of 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 2. Then, with the sorted actors, we use $\hat{W}^{\text{PE}}$ as a 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 $O(|A| \log |A|)$ and the complexity of the binary search is $O(\log |A|)$, the complexity of Algorithm 2 is $O(|A| \log |A|)$.

The processor type assignment can: (1) assign actors of an application CSDF 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 A^{EE} and A^{PE} returned by Algorithm 2, respectively. If $U^{\text{EE}} > M_c^{\text{EE}} \times M_p^{\text{EE}}$, the tasks from A^{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

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

Table 4.6: Different mappings for H.263 decoder

that the tasks can run on the system. Based on Proposition 1, it is trivial to observe that reassigning some of the tasks in set A^{EE} to set A^{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., $U^{\text{PE}} > M_c^{\text{PE}} \times M_p^{\text{PE}}$, that means that with the throughput and latency constraints the application is not schedulable on the system.

4.3.2 Task mapping

When the processor type assignment is determined as described in Section 4.3.1, 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 [GJ79]. Several well-known heuristic algorithms for the bin-packing problem, e.g., first-fit, best-fit, etc, have been proposed see Section 2.2.3. In terms of energy efficiency, the worst-fit-decreasing (WFD) algorithm is evaluated as the best mapping heuristic [AY03] for the partitioned scheduling. [KYD11] 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 4.6 shows two mappings for the H.263 decoder in Figure 4.4 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 VFS 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 4.3.1. 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:

$$f_j = \max\left(\max_{\forall \tau_i \in A_j}\{u_i\}, \frac{U_j}{M_p}\right) \times f_{\max} \tag{4.7}$$

where u_i is the utilization of task $\tau_i \in A_j$ and A_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 A_j and M_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 Equation (4.7).

With Equation (4.7), 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 4.3.3.

Definition 4.3.1. An U-cluster is a cluster where $\max_{\forall \tau_i \in A_j}\{u_i\} < \frac{U_j}{M_p}$.

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

Definition 4.3.2. A T-cluster is a cluster where $\max_{\forall \tau_i \in A_j}\{u_i\} \geq \frac{U_j}{M_p}$.

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

4.3.3 Remapping

The FFD algorithm mentioned in Section 4.3.2 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 by using VFS. 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 4.3.2. Below, we discuss how to remap tasks for both categories of clusters in order to reduce the energy consumption.

U-cluster

According to Definition 4.3.1, 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 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:

$$E = hp\big(U_1\alpha f_1^{(b-1)} f_1^{\max} + M_p\beta + P_s(f_1) + U_2\alpha f_2^{(b-1)} f_2^{\max} + M_p\beta + P_s(f_2)\big) \quad (4.8)$$

where f_1 is the operating frequency of the initial cluster, and f_2 is the operating frequency of the new cluster. In Equation (4.8), 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 Equation (4.8). 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 3 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 3 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

Algorithm 3: Split tasks for U cluster

Input: A task set Γ
Output: two tasksets Γ_1 and Γ_2

```
Sort tasks in Γ in order of decreasing utilization;
Γ1 ← Γ,   Γ2 ← ∅;
for i = 1 to |Γ| do
    if U2 ≤ U1 then
        Γ1 ← Γ1 − τi;
        Γ2 ← Γ2 + τi;
    else
        Break;
return Γ1 and Γ2;
```

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:*

$$\begin{aligned}&\left(U_1\alpha(f^{(b-1)} - f_1^{(b-1)}) + U_2\alpha(f^{(b-1)} - f_2^{(b-1)})\right)f^{\max} \\ >&P_s(f_1) + P_s(f_2) + M_p\beta - P_s(f)\end{aligned} \tag{4.9}$$

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. $f^{\max}$ is the maximum operating frequency of the cluster.[1]

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

$$E = hp\big(U\alpha f^{(b-1)} f^{\max} + M_p\beta + P_s(f)\big) \tag{4.10}$$

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

$$E_n = hp\big(U_1\alpha f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1) + U_2\alpha f_2^{(b-1)} f^{\max} + M_p\beta + P_s(f_2)\big) \tag{4.11}$$

[1] Since we only do remapping on the same type of cluster, the maximum operating frequency is the same.

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 Equation (4.10) and (4.11) in the inequality and eliminating the same terms on both sides, we obtain the following condition:

$$\begin{aligned} &\left(U_1\alpha(f^{(b-1)} - f_1^{(b-1)}) + U_2\alpha(f^{(b-1)} - f_2^{(b-1)})\right) f^{\max} \\ >&P_s(f_1) + P_s(f_2) + M_p\beta - P_s(f) \end{aligned} \tag{4.12}$$

□

T-cluster

According to Definition 4.3.2, in a T-cluster, the frequency of the cluster is determined by the **constrained task** (Definition 4.3.3). However, within one cluster, the FFD discussed in Section 4.3.2 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 4.3.2. 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 4.3.2 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 4 presents the pseudo-code of splitting tasks of a T-cluster. The complexity of Algorithm 4 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*

Algorithm 4: Split tasks for T-cluster

Input: A task set Γ
Output: two tasksets $\Gamma 1$ and Γ_2

```
Sort tasks in Γ in order of decreasing utilization;
Γ_1 ← ∅,   Γ_2 ← ∅;
for i = 1 to |Γ| do
    if τ_i can run at a lower frequency then
        for j = i to |Γ| do
            Γ_1 ← Γ_1 + τ_j;
        Γ_2 ← Γ − Γ_1;
        Break;
return Γ_1 and Γ_2;
```

taskset Γ_1 to an unused cluster can reduce the energy consumption, if the following condition is met:

$$U_1 \cdot \alpha \cdot f^{(b-1)} f^{\max} > U_1 \cdot \alpha \cdot f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1) \tag{4.13}$$

where f and f_1 are the operating frequencies of the initial and new cluster, respectively. $f^{\max}$ is the maximum operating frequency of the cluster. [2]

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:

$$E = hp\big(U \cdot \alpha \cdot f^{(b-1)} f^{\max} + M_p\beta + P_s(f)\big) \tag{4.14}$$

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:

$$E_n = hp\big(U_2 \cdot \alpha \cdot f^{(b-1)} f^{\max} + M_p\beta + P_s(f) + U_1 \cdot \alpha \cdot f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1)\big) \tag{4.15}$$

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 Equation (4.15). By substituting Equation (4.14) and (4.15) and eliminating the same terms on both sides of inequality $E > E_n$, we obtain:

$$U_1 \cdot \alpha \cdot f^{(b-1)} f^{\max} > U_1 \cdot \alpha \cdot f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1) \tag{4.16}$$

[2] Same type of clusters has the same maximum operating frequency.

□

4.3.4 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 $G = (A, \mathcal{E})$ 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 5 shows the pseudo-code of FDM. In Line 1, FDM applies Algorithm 2 explained in Section 4.3.1 to split tasks into two sets A^{PE} and A^{EE} which denote set of the tasks assigned to PE type and EE type of processors, respectively. In Algorithm 2, the processor type assignment completes the assignment procedure with the guarantees of meeting the performance constraints. Hence, if the task sets A^{EE} and A^{PE} derived by Algorithm 2 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 A^{EE} and assign them to set A^{PE} such that we have enough EE processors to schedule the tasks in set A^{EE}. The tasks are selected in order of decreasing utilization, and the selection is terminated as soon as the tasks in set A^{EE} are schedulable on the EE processors. However, if there is no enough PE type of processors, this 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 4.3.2 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 set of active clusters, 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 Equation (4.7).

With the obtained initial mapping, a remapping procedure starts from Line 7. In this procedure, we go through every cluster in 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 3 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 4 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 Equation (4.7). At Line 23, FDM outputs the final mapping and the operating frequency of each active cluster. Since the complexity of Algorithm 3 and 4 are both $O(|A|\log(|A|))$, in the worst case the complexity of FDM is $O(N \times |A|\log(|A|))$, where N is the total number of clusters in the input MPSoC and $|A|$ is the total number of actors in the input CSDF graph.

4.4 Evaluation

In this section, we present three experiments to demonstrate the efficiency of the proposed FDM algorithm compared to the existing approaches proposed in [CKR14] and [KYD11]. We apply our FDM and the mapping approaches from [CKR14] and [KYD11] on cluster heterogeneous and homogeneous MPSoCs. We choose [CKR14] and [KYD11] to compare with, because the mapping approaches in [CKR14] and [KYD11] are specifically devised for cluster MPSoCs as considered in our work. Therefore, their work is the most related and relevant to our approach.

We select 11 real-life streaming applications from the StreamIt [TA10] benchmark suite and the MP3 decoder [SGB06], where all streaming applications are modeled as CSDF graphs. We use the same parameters, i.e., WCETs of application tasks, as specified in [BS11]. An overview of all streaming applications is given in Table 4.7. $|A|$ denotes the number of tasks/actors in a CSDF graph, while $|\mathcal{E}|$ denotes the number of edges. L is the minimum achievable latency and $\mathcal{R}$ is the maximum achievable throughput which are computed by using Equation (2.11) and Equation (2.12) in Section 2.3, when the applications are scheduled by the HRT scheduling. 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 4.7) 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 4.8. 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 heterogeneous MPSoC, we use the energy model described in Section 4.2.2, where the model parameters are given in Table 4.3 and Table 4.2. In our experiments, we use our FDM approach and the reference mapping approaches described in [CKR14] and [KYD11] to map the tasks of the streaming applications to the three MPSoCs and we compute the energy consumption of each application to

Algorithm 5: Frequency Driven Mapping

Input: A CSDF graph $G = (A, \mathcal{E})$ and a cluster heterogeneous MPSoC
Output: A task mapping for each cluster and the minimum operating frequency for each active cluster

$A^{\text{PE}}, A^{\text{EE}} \leftarrow$Apply Algorithm 2 to split all actors $\tau_i \in A$ to two parts;
if $\lceil U^{EE} \rceil > M_c^{EE} \times M_p^{EE}$ **then**
    Map some actors τ_i to PE clusters in order of decreasing utilization such that $\lceil U^{\text{EE}} \rceil \leq M_c^{\text{EE}} \times M_p^{\text{EE}}$;
if $\lceil U^{PE} \rceil > M_c^{PE} \times M_p^{PE}$ **then**
    return Unschedulable;
$\Phi_{ac} \leftarrow$Apply FFD on PE clusters and EE clusters to generate an initial task mapping and compute the frequency of each active cluster by using Equation (4.7);
`/* Remapping procedure */`
for *j=1 to* $|\phi_{ac}|$ **do**
    if $\max_{\forall \tau_i \in A_j}\{u_i\} < \frac{U_j}{M_p}$ *& an unused cluster of the same type is available* **then**
        `// U-cluster`
        $A_{j,1}, A_{j,2} \leftarrow$ Apply Algorithm 3 to split tasks;
        if $A_{j,1}$ *and* $A_{j,2}$ *can meet the condition in Proposition 2* **then**
            Keep $\forall \tau_i \in A_{j,1}$ on the initial cluster and remap $\forall \tau_i \in A_{j,2}$ to unused cluster ϕ_{unused};
            $\Phi_{ac} \leftarrow \Phi_{ac} + \phi_{\text{unused}}$;
        else
            Keep the initial mapping;
    if $\max_{\forall \tau_i \in A_j}\{u_i\} \geq \frac{U_j}{M_p}$ *& an unused cluster of the same type is available* **then**
        `// T-cluster`
        $A_{j,1}, A_{j,2} \leftarrow$ Apply Algorithm 4 to split tasks;
        if $A_{j,1}$ *can meet the condition in Proposition 3* **then**
            Keep $\forall \tau_i \in A_{j,2}$ on the initial cluster and remap $\forall \tau_i \in A_{j,1}$ to unused cluster ϕ_{unused};
            $\Phi_{ac} \leftarrow \Phi_{ac} + \phi_{\text{unused}}$;
        else
            Keep the initial mapping;
Update the operating frequencies of clusters in set Φ_{ac} by using Equation (4.7);
return Φ_{ac};

MPSoC mapping configuration using Equation (4.4), (4.5) and (4.6). 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

APP	$\|A\|$	$\|\mathcal{E}\|$	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

Table 4.7: The Streaming Applications

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

Table 4.8: Cluster Heterogeneous MPSoC configurations

equation to compute the energy reduction:

$$r = \frac{E_{\text{ref}} - E_{\text{FDM}}}{E_{\text{ref}}} \tag{4.17}$$

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 VFS.

Comparison with [CKR14] on Heterogeneous MPSoCs

In this section, we compare our proposed FDM approach to the mapping approach proposed in [CKR14]. In [CKR14], the authors proposed several mapping approaches for cluster heterogeneous MPSoCs, and in our experiments we select the best mapping approach evaluated in [CKR14] and refer to it as CKR. In this experiment, the CKR is considered as the reference point and the energy reduction for each application

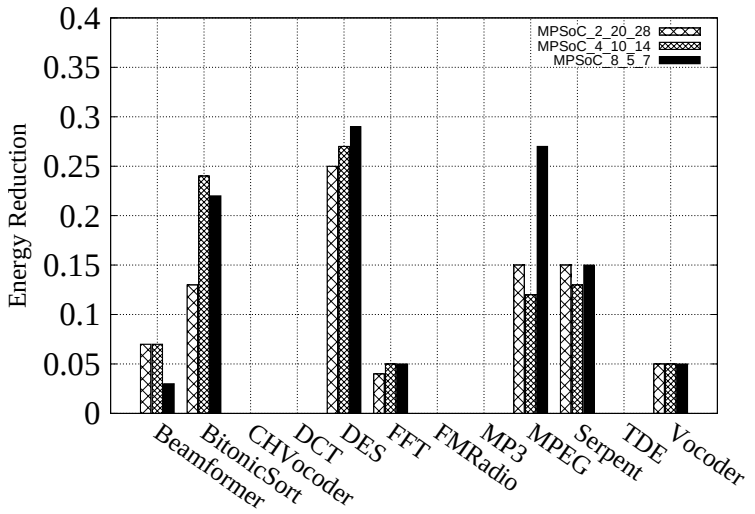

Figure 4.5: Comparison between FDM and CKR

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

Table 4.9: Summary of Figure 4.5

benchmark is computed by using Equation (4.17). Figure 4.5 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 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.

The energy reduction results are summarized in Table 4.9. 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.

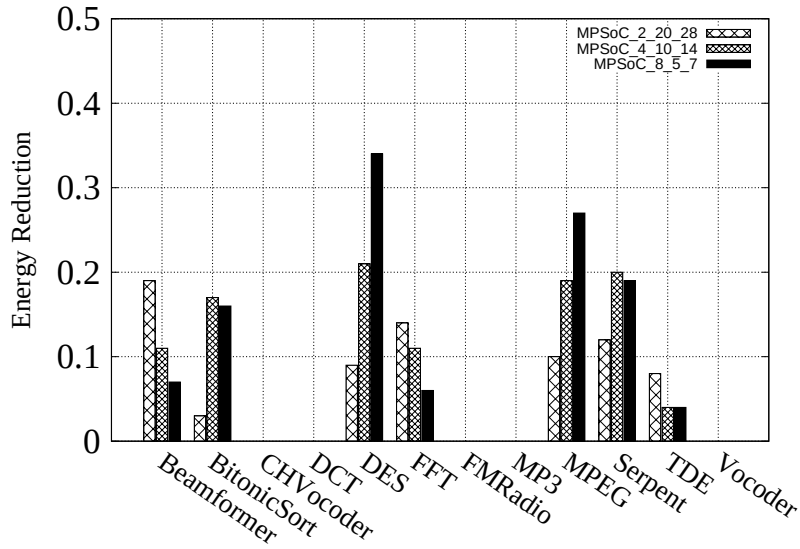

Figure 4.6: FDM vs. Algorithm 2+KYD

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

Table 4.10: Summary of Figure 4.6

Comparison with [KYD11] on Heterogeneous MPSoCs

In this experiment, we compare our FDM approach with the approach proposed in [KYD11] which we refer to as KYD. Since [KYD11] only considers cluster homogeneous MPSoCs, we apply our processor type assignment proposed in Section 4.3.1, i.e., Algorithm 2, 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 2+KYD' is used as the reference mapping approach in Equation (4.17).

The energy reduction for the different benchmarks mapped on the different MPSoCs is depicted in Figure 4.6. For 7 out of 12 benchmarks, our FDM finds a mapping which consumes less energy than the mapping approach 'Algorithm 2+KYD'. For the rest of the benchmarks, our proposed approach finds a mapping that consumes the same energy as the reference mapping approach. For benchmarks CHVocoder, 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,

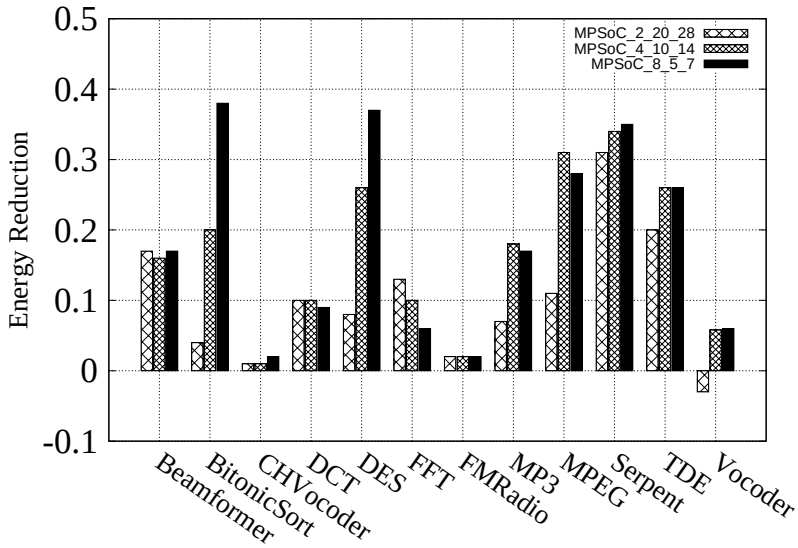

Figure 4.7: FDM vs. KYD on homogeneous MPSoCs

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

Table 4.11: Summary of Figure 4.7

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 4.10. 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.

Comparison with [KYD11] on Homogeneous MPSoCs

The KYD approach [KYD11] 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 4.7, 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 4.8, thereby considering cluster homogeneous MPSoCs that can meet the performance constraints for every application.

Figure 4.7 depicts the energy reduction of each benchmark mapped on the three different MPSoCs by only using the PE clusters. In Figure 4.7, 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 step 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 4.11. 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%.