Energy-Efficient Mapping of Real-Time Applications on Heterogeneous MPSoCs using Task Replication

Jelena Spasic Di Liu Todor Stefanov

Leiden Institute of Advanced Computer Science
Leiden University, Leiden, The Netherlands
Email: {j.spasic, d.liu, t.p.stefanov}@liacs.leidenuniv.nl

ABSTRACT

In this paper, we study the problem of exploiting parallelism in a hard real-time streaming application modeled as a Synchronous Data Flow (SDF) graph and scheduled on a cluster heterogeneous Multi-Processor System-on-Chip (MPSoC) platform such that energy consumption is minimized and a throughput requirement is satisfied. We propose a polynomial-time solution approach which: 1) determines a processor type for each task in an SDF graph such that the throughput constraint is met and energy consumption is minimized; 2) determines a replication factor for each task in an SDF graph such that the distribution of the workload on the same type of processors is balanced, which enables processors to run at a lower frequency, hence reducing the energy consumption. Experiments on a set of real-life streaming applications demonstrate that our approach reduces energy consumption by 66% on average while meeting the same throughput requirement when compared to related energy minimization approaches.

1. INTRODUCTION

Nowadays, embedded systems are built around a component which integrates multiple processors, memories, interconnects and other modules in a chip – so called Multi-Processor System-on-Chip (MPSoC). The growth of MPSoCs enables embedded systems to run very complex streaming applications which have high computational requirements and hard real-time constraints. Given that the embedded systems are very often battery-powered, another very important requirement in the design of embedded streaming MPSoCs is the energy-efficiency. Heterogeneous MPSoCs were identified as a promising solution in terms of energy-efficiency [19]. Especially, the *asymmetric multi-core* architecture, also known as *single-ISA heterogeneous* architecture, was recognized as a good trade-off in terms of energy-efficiency and programming effort [19]. A single-ISA heterogeneous MPSoC consists of cores with different power-performance characteristics but with the same instruction-set architecture (ISA). Apart from containing cores with different power-performance characteristics, such heterogeneous MPSoCs cover large set of power-performance design points through voltage-frequency scaling (VFS) of the cores [19]. However, with the advent of manycore systems, per-core VFS becomes impractical due to the high hardware cost and area requirement [11]. Therefore, to balance the energy saving and the hardware cost, cores are grouped into clusters and cores in each cluster run at the same voltage and frequency level. Some examples of commercial asymmetric cluster MPSoCs are Samsung Exynos 5 Octa SoC [24], nVidia Tegra X1 [21], which include ARM big.LITTLE [10] integrating high-performance cores into 'big' clusters and low-power cores into 'little' clusters.

In order to efficiently utilize such clustered heterogeneous platforms to achieve all the desired requirements, the underlying hardware platform and the running streaming application have to be closely related. This requires the embedded designer to expose the right amount of parallelism available in the application and to decide how to allocate and schedule the tasks of the application on the available processing elements such that the platform is utilized efficiently and the timing constraints are met. Several parallel Models-of-Computation (MoCs), e.g. Synchronous Data Flow (SDF) [14] and Cyclo-Static Dataflow (CSDF) [4], have been adopted as the parallel application specification. Within a parallel MoC, an application is represented as a task graph with concurrently executing and communicating tasks. Thus, the parallelism is explicitly specified in the model.

However, the given initial parallel application specification often is not the most suitable one for the given MPSoC platform. This is because application developers mainly focus on realizing certain application behavior while the computational capacity and power consumption profile of the MPSoC platform is often not fully taken into account. Hence, it may happen that an application consists of highly imbalanced tasks in terms of the task workload, i.e., task utilization. Especially, in cluster heterogeneous MPSoCs, when several tasks are mapped onto the same cluster, the one with the heaviest utilization will determine the required voltage and frequency of the whole cluster and will significantly increase the energy consumption of the other tasks mapped on the same cluster. When task replication is applied to application tasks with heavy utilization, their utilization can be decreased while still providing the same application performance. Thus, to better utilize the underlying MPSoC platform while minimizing the energy consumption, the initial specification of an application, i.e., the initial task graph, should be transformed to an alternative one that exposes more parallelism while preserving the same application behavior and timing performance. However, having more tasks' replicas than necessary introduces more overheads in code and data memory, scheduling and inter-tasks communication, which in turn will result in higher energy consumption. Thus, the right amount of parallelism (tasks' replicas), i.e., the proper values of replication (unfolding) factors, depending on the underlying MPSoC platform, should be determined in a parallel application specification to achieve the required performance and timing guarantees while minimizing the energy consumption.

Therefore, in this paper, we propose a novel algorithm to efficiently map real-time streaming applications onto cluster heterogeneous MPSoCs, which are subject to throughput constraints, such that the energy consumption of the cluster heterogeneous MPSoC is reduced by using task replication and per-cluster VFS. The specific novel contributions of this paper are the following:

CODES/ISSS '16, October 01-07, 2016, Pittsburgh, PA, USA

DOI: http://dx.doi.org/10.1145/2968456.2968474

- We propose a novel polynomial-time algorithm, called Data Parallel Energy Minimization (DPEM), to map and schedule hard real-time streaming applications onto a cluster heterogeneous MPSoC such that the energy consumption is minimized while the throughput constraints are guaranteed. By using the hard real-time scheduling of CSDF graphs in [26], we propose within our DPEM algorithm an efficient way to determine a suitable processor type for each task in an (C)SDF graph such that the energy consumption is minimized and the throughput constraint is met. Then, by using the unfolding graph transformation in [27], we propose a method in DPEM to determine a replication factor for each task in an SDF graph such that the distribution of the workload on the same type of processors is balanced, which enables processors to run at a lower frequency, hence reducing the energy consumption.
- We show, on a set of real-life streaming applications, that our proposed energy minimization approach outperforms related approaches in terms of energy consumption while meeting the same throughput constraints.

The remainder of the paper is organized as follows: Section 2 gives an overview of the related work. Section 3 gives a motivational example. Section 4 introduces the background necessary to understand the proposed energy minimization approach. The proposed approach is described in Section 5. Experimental evaluation of the approach is given in Section 6, and Section 7 concludes the paper.

2. RELATED WORK

Energy-efficient mapping and scheduling of streaming applications represented as dataflow graphs which guarantees certain throughput has been extensively studied. The related works can be divided into several categories depending on the MPSoC platform they consider: *homogeneous* [25, 8, 31, 16, 20, 5, 12], or *heterogeneous* [13, 23]. Depending on the VFS technique they apply to minimize the energy consumption, the related works can be divided into those considering *per-core VFS* [25, 8, 31, 16, 20, 13, 5], those considering *global VFS* [13, 12] and the works which do not consider VFS but they utilize platform heterogeneity to achieve energy-efficiency [23]. The approaches in [13, 16, 20, 23, 12] convert an initial SDF graph into equivalent Homogeneous SDF (HSDF) graph to exploit the parallelism of an application and achieve energy-efficiency. However, the HSDF graph obtained from the initial SDF graph may grow in size exponentially, making the analysis performed on the HSDF graph time-consuming. Instead, the approaches in [25, 31, 8, 5] perform energy minimization directly on an SDF graph. Works [25] and [31] perform design space exploration at design time to find the energy-efficient mapping solution of an SDF scheduled in self-timed manner on a homogeneous MPSoC platform with per-core VFS capability such that certain throughput is guaranteed. In addition, the approach in [25] has a run-time phase where slack created at run time is exploited to further minimize the energy consumption. In [8] the authors propose a heuristic to find per-core voltage-frequency points for a given task mapping and the execution order such that the throughput constraint is met. The authors in [5] propose a technique to transform an SDF graph at run time into its equivalent SDF graph to adapt to environmental and demand changes. One possible scenario where the SDF graph should be transformed to adapt to the new circumstances is when some processors become available on a homogeneous platform with per-core VFS capability. In that case the tasks in the SDF graph are replicated such that all processors are occupied, which enables processors to run at a lower frequency hence consuming less energy. However, the authors in [5] focus more on the transformation itself and not on the energy minimization. In contrast to all related works, discussed above, our approach: 1) considers heterogeneous MPSoC platforms with per-cluster VFS capability, which is a good trade-off in terms of energy-efficiency and the implementation cost; 2) utilizes an unfolding graph transformation to balance the workload put on the MPSoC and to reduce energy consumption by finding how many times each task in a graph should be replicated; 3) uses preemptive hard real-time scheduling to schedule the tasks which gives more opportunities to meet the lowest frequency for schedulability supported by the platform.

Energy-efficient mapping and scheduling of periodic hard real-time tasks has been widely researched in the past. [2] gives a comprehensive review of works dealing with energy-aware scheduling for real-time systems. As stated in [2], most of the existing work considers homogeneous MPSoCs and in recent years people started considering heterogeneous platforms and platforms with voltage/frequency levels shared among multiple processors as energy-efficient design solutions. Regarding the considered heterogeneous MPSoC platforms, the closest to our work are the works in [7] and [18]. The approach in [7] proposes and evaluates several partitioned Earliest Deadline First (EDF) scheduling strategies for real-time tasks mapped on cluster heterogeneous platforms in terms of energy-efficiency. However, because of the bin-packing issue in partitioned scheduling, the approach in [7] may not fully utilize the energy-efficient cores in a cluster heterogeneous MPSoC, hence the energy minimization is limited. In contrast, by replicating the tasks with heavy utilization, we can reduce their utilization and hence fully utilize the energy-efficient cores. The approach in [18] considers cluster scheduling for cluster heterogeneous MPSoCs where tasks are allowed to migrate at run-time among processors within the same cluster in order to achieve better resource utilization. However, cluster scheduling suffers from high scheduling overhead caused by task migration and increased context switching. Moreover, the frequency of some clusters in [18] is still determined by the tasks with the heaviest utilization. In contrast, in our approach, we use partitioned scheduling which has low scheduling overhead and we avoid the capacity loss and we lower the operating frequency by replicating the tasks with heavy utilizations.

The works in [30], [29] and [15] consider parallel execution of task replicas to achieve energy efficiency, as we do. The authors in [30] consider frame-based tasks with an implicit deadline and a homogeneous platform with per-core VFS capability where the frequency of a core may be changed for each task. In contrast, in our work, we consider more general periodic task model and more realistic heterogeneous platform with per-cluster VFS capability, hence our approach is more applicable in practice than the approach in [30]. The approach in [29] exploits the data parallelism in an application by replicating the tasks of the application over all processors available in an MPSoC. This means that, in distributed memory architectures, the code of the whole application has to be replicated on all the processors in an MPSoC. By contrast, in our approach, only certain tasks of the application have to be replicated, which reduces significantly the memory overhead of our approach compared to the one in [29]. Moreover, the work in [29] assumes homogeneous systems with per-core VFS and continuous frequencies, while we consider heterogeneous systems with per-cluster VFS capability, which is more practical in modern embedded systems. The approach presented in [15] replicates computation-intensive tasks which yields to a more balanced load on processors, and in turn allows the system to run at a lower frequency. In addition, the authors in [15] consider systems with discrete set of operating frequencies and homogeneous platforms with per-core VFS capability. As discussed earlier, per-core VFS is not practical in modern many-core systems. Hence, our work considers heterogeneous platforms with per-cluster VFS capability. The approach in [15] is devised and, hence efficient only for platforms with performance-efficient processors. This means that the approach in [15] would never replicate the tasks which are going to be mapped on energy-efficient processors. In addition, if the total number of tasks with heavy utilization is equal to the number of processors in a platform, tasks will not be replicated in [15]. In contrast, our approach will replicate the tasks mapped on energy-efficient processors and it will replicate the tasks even if the number of heavy

tasks is equal to the number of processors if this leads to more energy-efficient design.

3. MOTIVATIONAL EXAMPLE

In the first part of this section, we motivate the need for using the unfolding graph transformation to achieve energy-efficient MPSoC design under a throughput constraint. We first show the drawback of the energy minimization approaches for heterogeneous MPSoCs and hard real-time scheduling, i.e., the approaches in [7] and [18]. We analyze three different designs obtained by mapping the SDF graph G in Figure 1 to a heterogeneous platform consisting of one PE cluster with 2 PE processors and one EE cluster with 2 EE processors, i.e., the platform given in column 1, row 2 in Table 1, under a throughput constraint of 1 output token per $100\mu s$. The first design is obtained by using the best mapping approach evaluated in [7] and we refer to that approach as CKR. The CKR approach allocates actors τ_2 and τ_3 to PE processors in one-to-one manner, and it allocates actors τ_1 and τ_4 to one EE processor, while the other EE processor is switched-off. Once the actors are allocated, the minimum frequency which ensures the schedulability of actors mapped to a processor in a cluster is selected from the discrete set of frequencies per cluster. The energy consumption of such a design is given in Table 1, column 2, row 2. After applying the approach in [18], we obtain the second design where actors τ_2 and τ_3 are allocated to the PE cluster, while actors τ_1 and τ_4 are allocated to the EE cluster. The corresponding energy consumption after applying the approach in [18], denoted by FDM, is given in Table 1, column 3, row 2. If we apply our approach presented in Section 5 which uses the unfolding transformation in [27] on graph G in Figure 1, under the same throughput constraint as in the CKR and FDM approaches, we can lower the utilization of the actors with high utilization, τ_2 and τ_3, and achieve better load balancing on the processors of the same type and hence, the frequency of the power-hungry processors can be lowered further than in [7], [18]. For example, by unfolding actors τ_2 and τ_3 twice, as given in Figure 2, our approach in Section 5 allocates $\tau_{2,1}$ and $\tau_{3,1}$ one-to-one to PE processors, and it allocates $\tau_{2,2}$ to an EE processor and $\tau_{3,2}$, τ_1 and τ_4 to another EE processor. The energy consumption value for this third design is given in Table 1, column 5, row 2. We can see that our approach reduces the energy consumption by 71% when compared to the CKR and FDM approaches.

Now, we would like to analyze an approach which was devised for homogeneous platforms with per-core VFS capability, i.e., the approach in [25], denoted by SDK in Table 1. To this end, we compare the energy consumption of two designs in which the SDF graph G in Figure 1 is mapped to a homogeneous MPSoC consisting of four PE clusters with 1 processor per cluster, under a throughput constraint of 1 output token per $100\mu s$. The SDK approach will allocate actors to processors in one-to-one manner, while our approach, proposed in Section 5, will replicate actors τ_2 and τ_3 twice, as shown in Figure 2, to lower their utilization. We can see from columns 4 and 5, row 3 in Table 1 that our approach reduces the energy consumption by 50% when compared to the SDK approach. The main reason is that we are using the unfolding graph transformation to reduce the influence of heavy actors and hence minimize the energy of an MPSoC. Another reason is that the SDK approach uses self-timed scheduling which is non-preemptive, hence, less flexible for scheduling and that the SDK only minimizes dynamic energy consumption. The energy consumption values of the approaches CKR, FDM and SDK in Table 1 which were not discussed above are given only for completeness. However, we can see that these values are always higher than the corresponding energy consumption of our approach in Section 5. Thus, our approach outperforms these related approaches.

Above, we motivated the need to use the unfolding transformation within our new approach in Section 5 to achieve energy-efficiency for MPSoCs under a throughput constraint. Now, we would like to motivate the need for our whole approach, presented in Section 5,

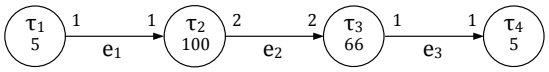


Figure 1: An SDF graph G.

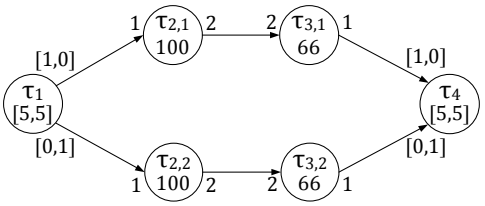


Figure 2: A CSDF graph G' obtained by unfolding SDF graph G in Figure 1 with $\vec{f} = [1, 2, 2, 1]$.

Table 1: Different MPSoC designs for G in Figure 1.

MPSoC	CKR [μJ]	FDM [μJ]	SDK [μJ]	our [μJ]	WYL [μJ]
(2PE)(2EE)	343.55	343.55	346.30	97.76	343.55
(PE)(PE)(PE)(PE)	357.94	392.18	389.09	192.80	240.32

which efficiently finds task replication factors and task mappings to achieve further reductions in the energy consumption. Although the approach in [15] exploits the energy-saving capability of data-parallel execution for homogeneous MPSoC with per-core VFS capability, that approach is not efficient in terms of energy reduction, especially in the case of platforms with EE processors. Below we show its inefficiency for the homogeneous platform in column 1, row 3, and for the heterogeneous MPSoC platform in column 1, row 2, in Table 1. The approach in [15], called the WYL approach, considers platforms which power consumption curve is increasing 'fast' with the increase of processor utilization. Such power consumption curve corresponds to PE processors. Let us consider the mapping of the SDF graph in Figure 1 on the homogeneous platform under a throughput constraint of 1 output token per $100\mu s$. The WYL approach will classify actors τ_2 and τ_3 as 'heavy' tasks, i.e., tasks eligible for replication. However, because the platform contains only 4 processors, the WYL will decide that the number of processors is not sufficient to replicate both actors and it will only replicate actor τ_2 twice. In contrast, our algorithm will replicate both actors τ_2 and τ_3 twice, which will lead to an energy reduction of 20%, see Table 1, row 3, columns 5 and 6.

Let us now analyze the designs obtained by applying the WYL and our new approach for mapping graph G in Figure 1 on the heterogeneous platform in column 1, row 2, in Table 1. Given that the power consumption curve of EE processors is a 'slowly' increasing curve with the increase of processor utilization, the WYL approach will never replicate actors assigned to EE processors. In contrast, our approach presented in Section 5 will replicate actors assigned to EE processors as well if their replication leads to more energy-efficient MPSoC. We can see in row 2, columns 5 and 6, in Table 1, that our approach leads to a design with 71% reduction in energy consumption when compared to the WYL approach, for the heterogeneous MPSoC with one PE and one EE cluster each containing 2 processors. This happens because after the classifications of actors into PE and EE in order to satisfy the throughput constraint of 1 output token per $100\mu s$, EE actors will not be considered for replication in the WYL approach, while PE actors will be considered yet never replicated because of the algorithm in [15] which does not replicate the actors once the number of 'heavy' actors is equal to the number of (PE) cores, which happens for this platform.

From the above examples, we can see the necessity and usefulness of our approach, presented in Section 5, which uses the graph unfolding transformation to obtain energy-efficient cluster heterogeneous MPSoC designs.

4. BACKGROUND

Given that the unfolding transformation, we use to replicate the tasks in an application modeled as an SDF graph, transforms the graph into a CSDF graph and that the CSDF MoC is a superset of the SDF MoC, in this section, we first introduce the CSDF MoC, followed by the unfolding transformation. Then, we review the scheduling framework proposed in [26], which we use to schedule tasks in a CSDF graph. After that we present the system model and energy model considered in this paper.

4.1 Cyclo-Static Dataflow (CSDF)

An application modeled as a CSDF [4] is a directed graph $G = (V, E)$ that consists of a set of actors V which communicate with each other through a set of communication channels E. Actors represent a certain functionality of the application, while communication channels are FIFOs representing data dependency and transferring data tokens between the actors. Every actor $\tau_i \in V$ has an *execution sequence* $[f_i(1), f_i(2), \cdots, f_i(P_i)]$ of length P_i, i.e., it has P_i phases. The kth time that actor τ_i is fired, it executes the function $f_i(((k-1) \bmod P_i) + 1)$. As a consequence, the execution time of actor τ_i is also a sequence $[C_i^C(1), C_i^C(2), \cdots, C_i^C(P_i)]$ consisting of the worst-case computation time values for each phase. Every output channel e_u of an actor τ_i has a predefined token *production sequence* $[x_i^u(1), x_i^u(2), \cdots, x_i^u(P_i)]$ of length P_i. Analogously, token consumption on every input channel e_u of an actor τ_i is a predefined sequence $[y_i^u(1), y_i^u(2), \cdots, y_i^u(P_i)]$, called *consumption sequence*.

An important property of the CSDF model is the ability to derive at design-time a schedule for the actors. In order to derive a valid static schedule for a CSDF graph at design-time, it has to be consistent and live. A CSDF graph G is said to be consistent if a positive integer solution $\vec{r} = [r_1, r_2, \cdots, r_N]^T$ exists for the *balance equation* $\mathbf{\Gamma} \cdot \vec{r} = \vec{0}$ in [4], where $\mathbf{\Gamma}$ represents the topology matrix containing for each $e_u = (\tau_i, \tau_j)$ in G the number of produced and consumed tokens during P_i and P_j firings of actors τ_i and τ_j, respectively. If a deadlock-free schedule can be found, G is said to be live. Each consistent CSDF graph has a non-trivial *repetition vector* $\vec{q} = [q_1, q_2, \cdots, q_N]^T \in \mathbb{N}^N$. An entry $q_i \in \vec{q}$ represents the number of invocations of an actor τ_i in a graph iteration of G. Similarly, an entry $r_i \in \vec{r}$ represents the number of invocations of a phase of an actor τ_i in a graph iteration of G. The smallest non-trivial repetition vector is called *basic repetition vector*. If every actor τ_i in a CSDF graph G has $P_i = 1$ then the graph G is an SDF graph, i.e., the SDF MoC is a subset of the CSDF MoC.

Figure 1 shows an example of an SDF graph. For instance, actor τ_2 has worst-case computation time $C_2^C = 100$ time units and its token production rate x_2^2 on channel e_2 is 2. The repetition vector of G in Figure 1 is $\vec{q} = [1, 1, 1, 1]^T$ and it is the basic repetition vector for G. Figure 2 shows an example of a CSDF graph. For graph G' shown in Figure 2, the basic repetition vector $\vec{q}$ is $[2, 1, 1, 1, 1, 2]^T$ and its corresponding vector $\vec{r}$ is $[1, 1, 1, 1, 1, 1]^T$. Throughout this paper, all SDF and CSDF graphs are assumed to be consistent and live.

4.2 Unfolding Transformation of SDF Graphs

The authors in [27] showed that an SDF graph can be efficiently transformed into an equivalent CSDF graph by graph unfolding to expose the right amount of parallelism in the SDF graph needed to better utilize the underlying MPSoC platform. The motivation behind the unfolding is to equally distribute the workload of an actor in the initial graph by running in parallel replicas corresponding to that actor. Given a vector $\vec{f} \in \mathbb{N}^N$ of unfolding factors, where f_i denotes the unfolding factor for actor τ_i, the unfolding transformation replaces τ_i with f_i replicas of τ_i. To ensure the functional equivalence, the production and consumption sequences on channels in the resulting CSDF graph are modified accordingly to the production and consumption rates in the initial SDF graph. After the unfolding, each replica $\tau_{i,k} \in G'$, $k \in [1, f_i]$, of an actor $\tau_i \in G$ will have the repetition $q_{i,k}$ [27]:

$$q_{i,k} = \frac{q_i \cdot \text{lcm}(\vec{f})}{f_i}, \tag{1}$$

where $\text{lcm}(\vec{f})$ is the least common multiple of all unfolding factors in $\vec{f}$. For example, after the unfolding of the SDF graph in Figure 1 with the unfolding vector $\vec{f} = [1, 2, 2, 1]$ we obtain the CSDF graph shown in Figure 2 with the repetition vector $\vec{q'} = [2, 1, 1, 1, 1, 2]^T$, where $q_{2,1} = q_{2,2} = \frac{1 \cdot lcm(1,2,2,1)}{2} = 1$.

4.3 Hard Real-Time Scheduling of (C)SDF

In [26], a real-time improved strictly periodic scheduling (ISPS) framework for acyclic CSDF graphs is proposed. In the framework in [26], every actor in a CSDF graph is converted to a set of *implicit-deadline periodic* (IDP) tasks. The analysis in [26] begins with the computation of the worst-case execution time (WCET) sequence $C_i = [C_i(1), C_i(2), \cdots, C_i(P_i)]$ of a CSDF actor τ_i. The WCET value $C_i(\varphi)$ for a phase φ is computed such that both the worst-case communication and computation times of a phase φ of τ_i is included:

$$C_i(\varphi) = C^R \cdot \sum_{e_r \in \text{in}(\tau_i)} y_i^r(\varphi) + C_i^C(\varphi) + C^W \cdot \sum_{e_w \in \text{out}(\tau_i)} x_i^w(\varphi), \tag{2}$$

where C^R represents the platform-dependent worst-case time needed to read a single token from an input channel e_r from the set of input channels $\text{in}(\tau_i)$ of actor τ_i; analogously, C^W is the worst-case time needed to write a single token to an output channel e_w from the set of output channels $\text{out}(\tau_i)$ of τ_i; $y_i^r(\varphi)$ and $x_i^w(\varphi)$ is the number of tokens read from e_r and written to e_w by τ_i, respectively, during its execution phase φ; and $C_i^C(\varphi)$ is the worst-case computation time of τ_i in its phase φ. To simplify the notations, the following definition is used:

Definition 1. The **workload** of an actor τ_i in a CSDF graph G is $W_i = \sum_{\varphi=1}^{\varphi=P_i} C_i(\varphi) \cdot r_i$ and the **maximum actor workload** of the graph G is $\hat{W} = \max_{\tau_i \in G}\{W_i\}$.

Every actor τ_i in a CSDF graph G is converted to a set of IDP tasks consisting of P_i tasks $\tau_i(\varphi) = (S_i(\varphi), C_i(\varphi), D_i, T_i)$, $1 \le \varphi \le P_i$, by computing the task parameters. Parameter $S_i(\varphi)$ is the earliest start time of a phase φ of actor τ_i, $C_i(\varphi)$ is the WCET value of a phase φ given by Eq. (2), D_i is the implicit deadline, and T_i is the period of tasks, where $T_i \ge \sum_{\varphi=1}^{\varphi=P_i} C_i(\varphi)$).

To execute graph G in strictly periodic fashion, period T_i for each actor τ_i is computed in [26] as follows:

$$T_i = \frac{\text{lcm}(\vec{r})}{r_i} \cdot s, \forall \tau_i \in V, \tag{3}$$

$$s = \left\lceil \frac{\hat{W}}{\text{lcm}(\vec{r})} \right\rceil, \tag{4}$$

where $\text{lcm}(\vec{r})$ is the least common multiple of all repetition entries in $\vec{r}$ and $\hat{W}$ is given by Definition 1. These derived actor periods ensure that each actor τ_i executes q_i times in every graph *iteration period* T_G, also called *hyperperiod*. Hyperperiod T_G for graph G is given by:

$$r_1 T_1 = r_2 T_2 = \cdots = r_{N-1} T_{N-1} = r_N T_N = T_G. \tag{5}$$

Note that periods computed by Eq. (3) are the minimum periods for actors scheduled by ISPS and that there exist other larger valid periods for actors by taking any integer $s > \left\lceil \frac{\hat{W}}{\text{lcm}(\vec{r})} \right\rceil$. Once the actor periods are computed, the utilization of actor τ_i, denoted as u_i, can be computed as $u_i = \sum_{\varphi=1}^{\varphi=P_i} C_i(\varphi)/T_i$, where $u_i \in (0, 1]$. For a graph

G, u_G is the total utilization of G given by:

$$u_G = \sum_{\tau_i \in V} u_i = \sum_{\tau_i \in V} \frac{\sum_{\varphi=1}^{\varphi=P_i} C_i(\varphi)}{T_i}. \quad (6)$$

The total utilization u_G of a graph G directly determines the minimum number of processors needed to schedule the graph.

The throughput of each actor τ_i can be computed as P_i/T_i. The throughput of a graph G when its actors are scheduled as strictly periodic tasks is determined by the period of the output actor and is given by:

$$\mathcal{R} = \frac{P_{out}}{T_{out}}. \quad (7)$$

The authors in [26] also provide a method for calculating the latency of a CSDF graph scheduled in a strictly periodic fashion. In addition, the framework computes the minimum buffer size for each channel in a graph such that actor phases, i.e., tasks, can be executed in strictly periodic fashion.

Converting the actors to periodic tasks, as described above, enables the application of the well-developed hard real-time scheduling theory [9], and hence, allows fast analytical calculation of the minimum number of processors needed to schedule the tasks in a CSDF. In real-time systems, tasks can be scheduled on processors by using global, hybrid, or partitioned scheduling algorithms [9]. However, global and hybrid scheduling algorithms require task migration, and thus introduce additional run-time overheads and memory overhead on distributed memory systems. The other class of scheduling algorithms are partitioned algorithms which do not require task migration, hence they have low run-time overheads. In partitioned scheduling, tasks are first allocated to processors and then scheduled on each processor by an uniprocessor scheduling algorithm. By performing extensive empirical comparison of global, clustered (hybrid) and partitioned algorithms for Earliest Deadline First (EDF) scheduling, the authors in [3] concluded that the partitioned algorithm outperforms all the algorithms when hard real-time systems are considered. Thus, in this paper, we consider partitioned scheduling algorithms.

4.4 System Model

We consider a cluster heterogeneous MPSoC containing two types of clusters – performance-efficient (PE) clusters and energy-efficient (EE) clusters. Each cluster has a number of identical PE processors, denoted as N_p^{PE}, or a number of EE processors, denoted as N_p^{EE}. Thus, in total, a cluster heterogeneous MPSoC contains $N_c^{PE} \times N_p^{PE}$ PE processors and $N_c^{EE} \times N_p^{EE}$ EE processors, where N_c^{PE} and N_c^{EE} represent the total number of PE clusters and the total number of EE clusters, respectively. All processors on the same cluster operate at the same voltage and frequency level. The voltage and frequency level of a cluster can be changed to control the power consumption. A cluster can be switched-off, thereby consuming no power.

Since actors may run on two different types of processors (PE and EE), the worst-case execution time value $C_i(\varphi)$ for each phase φ of an actor τ_i has two values – $C_i^{PE}(\varphi)$ and $C_i^{EE}(\varphi)$. The total utilizations of the tasks assigned to PE cluster j and EE cluster k can be calculated by:

$$u_j^{PE} = \sum_{\tau_i \in V_j^{PE}} \frac{\sum_{\varphi=1}^{\varphi=P_i} C_i^{PE}(\varphi)}{T_i}, \quad u_k^{EE} = \sum_{\tau_i \in V_k^{EE}} \frac{\sum_{\varphi=1}^{\varphi=P_i} C_i^{EE}(\varphi)}{T_i}, \quad (8)$$

where V_j^{PE} and V_k^{EE} represent sets of CSDF tasks assigned to PE cluster j and EE cluster k, respectively.

4.5 Energy Model

Given that all processors in the same cluster operate at the same voltage and frequency level, we can reduce the energy consumption of a cluster heterogeneous MPSoC by using per-cluster VFS and by switching-off some clusters. The authors in [18] give the power model for cluster heterogeneous MPSoC systems with discrete voltage and frequency levels based on real measurements performed on the ODROID XU-3 [22] board containing an MPSoC with two clusters – one quad core Cortex A15 'big' (PE) cluster and one quad core Cortex A7 'little' (EE) cluster. The power model of a cluster is given by:

$$P(f) = \alpha f^b + \beta N_{p,ac} + P_s(f), \quad (9)$$

where the first term is the dynamic power consumption, β is the static power consumption of one processor and $N_{p,ac}$ is the number of active processors on the cluster, $P_s(f)$ is the 'uncore' power consumption and f is the frequency level. 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. Parameters α, b and β, and $P_s(f)$ depend on the platform and cluster type, and they are determined in [18].

We calculate the total energy consumption for a graph G mapped onto a cluster heterogeneous MPSoC over one hyperperiod T_G by:

$$E = E^{PE} + E^{EE}. \quad (10)$$

E^{PE} in Eq. (10) contains the total energy consumption of PE clusters and is given by:

$$E^{PE} = T_G\Big(\sum_{j=1}^{N_{ac}^{PE}} \big(u_j^{PE}\alpha^{PE}(f_j)^{b^{PE}} + \beta^{PE}N_{p,ac_j}^{PE} + P_s^{PE}(f_j)\big)\Big), \quad (11)$$

where N_{ac}^{PE} is the number of active PE clusters, N_{p,ac_j}^{PE} is the number of active processors on PE cluster j, u_j^{PE} is the total utilization for tasks successfully scheduled by a partitioned scheduling algorithm on the corresponding PE cluster j, f_j is the operating frequency for the corresponding PE cluster j, and α^{PE}, b^{PE} and β^{PE} are the power parameters for PE clusters [18].

The total energy consumption of EE clusters, E^{EE} in Eq. (10), is given by:

$$E^{EE} = T_G\Big(\sum_{k=1}^{N_{ac}^{EE}} \big(u_k^{EE}\alpha^{EE}(f_k)^{b^{EE}} + \beta^{EE}N_{p,ac_k}^{EE} + P_s^{EE}(f_k)\big)\Big), \quad (12)$$

where N_{ac}^{EE} is the number of active PE clusters, N_{p,ac_k}^{EE} is the number of active processors on EE cluster k, u_k^{EE} is the total utilization for tasks successfully scheduled by a partitioned scheduling algorithm on EE cluster k, f_k is the operating frequency for the corresponding EE cluster k, and α^{EE}, b^{EE} and β^{EE} are the power parameters for EE clusters [18].

5. THE PROPOSED ENERGY MINIMIZATION APPROACH

In this section, we present our novel energy minimization approach called Data-Parallel Energy Minimization (DPEM) which energy-efficiently exploits a given cluster heterogeneous MPSoC platform when mapping a hard real-time streaming application under a throughput constraint. The logic behind our energy minimization approach is the following: our approach replicates the tasks with heavy utilization to reduce their utilization and lower the operating frequency, thereby reducing the energy consumption; it tries to map as many tasks as possible to EE processors such that the energy consumption is further reduced, while the throughput constraint is met. The DPEM approach is given in Algorithm 1 and explained in Section 5.1 while its constituents are desribed in Section 5.2 and Section 5.3.

5.1 The Data-Parallel Energy Minimization Algorithm

In this section, we present our integral algorithm for Data-Parallel Energy Minimization (DPEM). The inputs to DPEM are an SDF

Algorithm 1: Data-Parallel Energy Minimization (DPEM).

Input: An SDF graph $G = (V, E)$, a cluster heterogeneous MPSoC and a throughput constraint $\mathcal{R}$.

Output: Vector of unfolding factors $\vec{f}_{best}$, task mapping to processors in the clusters C_{best}, vector of operating frequencies for clusters $\vec{F}_{best}$ and the minimum energy consumption E_{best}.

1 $\vec{f} = [1, 1, \cdots, 1]$;
2 Calculate WCETs for each actor τ_i in G by using Eq. (2);
3 Calculate period T_i for PE type of processors for each actor τ_i in G by using Eq. (3) and $s = \left\lfloor \frac{P_{out} \cdot r_{out}}{\mathcal{R} \cdot \text{lcm}(\vec{r})} \right\rfloor$; $T_{best} = T_G = r_{out} \cdot T_{out}$;
4 Find the bottleneck actor $\tau_{b,k}$ in G;
5 V^{EE}, $V^{PE} \leftarrow$ Classify actors in G by Algorithm 2(G, $\frac{P_{out}}{T_{out}}$);
6 Find C_{best}, $\vec{F}_{best}$, E_{best} by Algorithm 3(V^{EE}, V^{PE});
7 **if** $C_{best} = \emptyset$ **then**
8 **return** Unschedulable;
9 **while** $f_b < (N_c^{EE} \times N_p^{EE} + N_c^{PE} \times N_p^{PE}) \wedge \tau_{b,k}$ *not stateful/in/out* **do**
10 $f_b = f_b + 1$;
11 Get G' by unfolding G using method in Section 4.2;
12 Calculate WCETs for each actor τ'_i in G' by using Eq. (2);
13 Calculate period T'_i for each actor τ'_i by using Eq. (3) and $s = \left\lfloor \frac{P'_{out} \cdot r'_{out}}{\mathcal{R} \cdot \text{lcm}(\vec{r'})} \right\rfloor$; $T_{G'} = r'_{out} \cdot T'_{out}$;
14 Find the bottleneck actor $\tau_{b,k}$ in G';
15 V'^{EE}, $V'^{PE} \leftarrow$ Classify actors in G' by Algorithm 2(G', $\frac{P'_{out}}{T'_{out}}$);
16 Find $C_{best,u}$, $\vec{F}_{best,u}$, $E_{best,u}$ by Algorithm 3(V'^{EE}, V'^{PE});
17 **if** $C_{best,u} = \emptyset$ **then**
18 **go to 9**;
19 **if** $\frac{\text{lcm}(T_{G'}, T_{best})}{T_{G'}} \cdot E_{best,u} < \frac{\text{lcm}(T_{G'}, T_{best})}{T_{best}} \cdot E_{best}$ **then**
20 $E_{best} = E_{best,u}$, $T_{best} = T_{G'}$, $\vec{F}_{best} = \vec{F}_{best,u}$, $C_{best} = C_{best,u}$, $\vec{f}_{best} = \vec{f}$;
21 **return** $\vec{f}_{best}$, C_{best}, $\vec{F}_{best}$, E_{best};

graph G, a cluster heterogeneous MPSoC, and a throughput constraint $\mathcal{R}$. The outputs are a vector of unfolding factors $\vec{f}_{best}$ according to which each actor in the initial SDF graph should be replicated, the task mapping to processors in the clusters C_{best}, a vector of operating frequencies for clusters $\vec{F}_{best}$ and the minimum energy consumption of the system E_{best}. The DPEM algorithm is shown in Algorithm 1. Line 1 in Algorithm 1 initializes each unfolding factor of an actor in graph G to 1. In Lines 2 and 3, the initial graph G is converted to periodic tasks by the ISPS approach in Section 4.3, where periods for each actor in G are set, by using scaling factor s in Line 3, to be as large as possible while meeting the throughput constraint $\mathcal{R}$. The corresponding hyperperiod T_G of graph G is calculated as well in Line 3. Line 4 finds the bottleneck actor in G. The bottleneck actor is the actor with the heaviest workload among the actor workloads for PE type of processors during one hyperperiod. If multiple actors have the same maximum workload, then the one with the smallest code size is selected to be the bottleneck. Note that stateful actors and input and output actors are not unfolded. In Line 5, Algorithm 2, explained in Section 5.2, is applied to classify actors into two groups – EE and PE. Here, by splitting actors into two groups, the required throughput of G under ISPS is guaranteed. Line 6 uses Algorithm 3, described in Section 5.3, to energy-efficiently map graph G on the input MPSoC platform. It may happen that the input platform is not big enough to map the input application, i.e., graph G. In that case Algorithm 3 will return an empty mapping, i.e., $C_{best} = \emptyset$. If this happens, the algorithm terminates and signals failure in Line 8. Otherwise, after obtaining the initial energy-efficient solution in Line 6, we further search to reduce the energy consumption by exploiting task replication via the unfolding, Lines 9 to 20.

Line 9 checks if the upper bound on the unfolding factor for the bottleneck actor has been reached and if the bottleneck actor is one of the actors which cannot be unfolded (input, output actors and stateful actors). If one of these happens, Algorithm 1 terminates and returns in Line 21 the most energy-efficient solution found so far. Otherwise, the initial SDF graph is transformed into an equivalent CSDF graph by replicating, in Line 10, the bottleneck actor previously found in Line 4. The graph transformation is performed in Line 11 by using the unfolding transformation method described in Section 4.2. Given that the transformed graph contains more actors than the original one, the WCETs of the actors have to be recomputed because the worst-case communication time may change. This is done in Line 12. Once the WCETs in the CSDF graph are recalculated, actors in the CSDF graph are transformed into periodic tasks by using the ISPS approach in Section 4.3. The unfolding graph transformation is usually used to increase the throughput of a graph by exposing more parallelism through task replication. However, here we want just to meet the same throughput constraint $\mathcal{R}$ as the initial graph, and use the unfolding transformation to change the utilization of the periodic tasks. To meet throughput constraint $\mathcal{R}$ and keep the throughput as close as possible to the initial throughput in Line 3, we scale the periods of the periodic tasks obtained after the conversion by scaling factor s, which is given in Line 13. Then, we find in Line 14 the bottleneck actor in the equivalent CSDF graph G', which is replicated in the next pass of the algorithm. The actors in G' are classified into PE and EE actors and the minimum energy of mapping the tasks corresponding to actors in G' onto the MPSoC is calculated in Lines 15 and 16. If there is no feasible mapping we continue with the task replication, Lines 17 and 18. On the other hand, if we could map G' on the MPSoC, the obtained energy is compared against the best, i.e., the minimum, energy obtained so far over the same time interval in Line 19. If we detect that the energy consumption of the current solution is smaller than the energy consumption of the best solution found so far, the current solution becomes the best one in Line 20. Line 9 checks whether the termination criteria for Algorithm 1 is met. If it is not, the algorithm will repeat Lines 10 to 20. Otherwise, the best solution is returned in Line 21.

Finally, we can analyze the time complexity of our DPEM algorithm in the worst case. The complexity of Algorithm 1 is determined by the **while loop** in Lines 9 to 20. In the worst case, the while loop will be executed until all the actors in the initial graph are replicated in the equivalent graph maximum number of times, which is equal to the number of processors N in the platform. So, the while loop will be executed $|V|N$ times in the worst case. The complexity of the graph unfolding algorithm in [27], which is called in Line 11, is $O(|E|N^2P)$, where P is the maximum number of execution phases per actor in the equivalent CSDF graph obtained after unfolding, i.e., $P = \max_{\tau_i \in V'}\{P_i\}$. The complexity of the other parts of the while loop is determined by Algorithm 3, see Section 5.3. Thus, the worst-case complexity of Algorithm 1 is $O(N|V| \cdot (N^2P|E| + (N|V|)^2 \log(N|V|)))$, which is polynomial.

5.2 Task Classification for Energy Minimization

In Algorithm 1, we used Algorithm 2 in Lines 5 and 15 to classify tasks of a graph into two groups, depending on the processor type they should be executed. Selecting the processor type to execute a task in an application is very important because different type of processors in a heterogeneous MPSoC have significantly different power and timing profiles. Algorithm 2 gives our task classification method. It takes a CSDF graph G and a throughput requirement $\frac{P_{out}}{T_{out}}$ as inputs and it produces PE and EE subsets of tasks in G.

First, we sort the tasks in order of increasing workload assuming all of them are assigned to EE processors – see Line 1 in Algorithm 2. Then, with the sorted tasks, we use the hyperperiod $r_{out} \cdot T_{out}$ as the classification threshold such that throughput requirement $\frac{P_{out}}{T_{out}}$ is met and the energy consumption is minimized, and deploy a binary search algorithm in Line 2 to find the pivotal point by which we can split the sorted tasks into two sets, one for the EE type of processor

Algorithm 2: Procedure to classify tasks according to processor type.

Input: A CSDF graph $G = (V, E)$ and a throughput constraint $\frac{P_{out}}{T_{out}}$.
Output: Subsets V^{PE} and $V^{EE} \subset V$.
1 $V \leftarrow$ Sort actors τ_i in V in increasing order of W_i^{EE};
2 $b \leftarrow$ Binary search to find the position in V with the biggest index where actor τ_i can meet $W_i^{EE} \leq r_{out} T_{out}$;
3 $V^{EE} \leftarrow V[0 : b]$;
4 $V^{PE} \leftarrow V - V^{EE}$;
5 **return** V^{EE}, V^{PE};

Algorithm 3: Procedure to find the minimum energy when the given tasks are mapped onto a cluster heterogeneous MPSoC.

Input: Sets of actors V^{EE} and V^{PE} and a cluster heterogeneous MPSoC.
Output: Task mapping to processor in the clusters C, vector of operating frequencies for clusters $\vec{F}$ and the minimum energy consumption E.
1 **if** *V^{EE} cannot be scheduled on $N_c^{EE} \times N_p^{EE}$ processors by WFD algorithm and max frequency f_{max}^{EE}* **then**
2 Move some actors $\tau_i \in V^{EE}$ to PE set V^{PE} in order of non-increasing u_i such that V^{EE} is schedulable on $N_c^{EE} \times N_p^{EE}$ processors;
3 **if** *V^{PE} cannot be scheduled on $N_c^{PE} \times N_p^{PE}$ processors by WFD algorithm and max frequency f_{max}^{PE}* **then**
4 **return** $C \leftarrow \emptyset$, $\vec{F} \leftarrow \emptyset$, $E = \infty$;
5 **if** $|V^{EE}| = 0$ **then**
6 $C^{EE} \leftarrow \emptyset$, $\vec{F}^{EE} \leftarrow \emptyset$, $E^{EE} = 0$;
7 **else**
8 $n_{lb}^{EE} = \left\lceil \frac{\lceil u^{EE} \rceil}{N_p^{EE}} \right\rceil$, $n_{ub}^{EE} = \min\{\left\lceil \frac{|V^{EE}|}{N_p^{EE}} \right\rceil, N_c^{EE}\}$;
9 Find C^{EE}, $\vec{F}^{EE}$, E^{EE} by Algorithm 4($n_{lb}^{EE}, n_{ub}^{EE}, V^{EE}$, Eq. (12));
10 **if** $|V^{PE}| = 0$ **then**
11 $C^{PE} \leftarrow \emptyset$, $\vec{F}^{PE} \leftarrow \emptyset$, $E^{PE} = 0$;
12 **else**
13 $n_{lb}^{PE} = \left\lceil \frac{\lceil u^{PE} \rceil}{N_p^{PE}} \right\rceil$, $n_{ub}^{PE} = \min\{\left\lceil \frac{|V^{PE}|}{N_p^{PE}} \right\rceil, N_c^{PE}\}$;
14 Find C^{PE}, $\vec{F}^{PE}$, E^{PE} by Algorithm 4($n_{lb}^{PE}, n_{ub}^{PE}, V^{PE}$, Eq. (11));
15 $C = \{C^{EE}, C^{PE}\}$, $\vec{F} = \{\vec{F}^{EE}, \vec{F}^{PE}\}$, $E = E^{EE} + E^{PE}$;
16 **return** C, $\vec{F}$, E;

and another for the PE type of processor. The goal is to put as many tasks as possible to EE processors to reduce the energy consumption while satisfying the throughput requirement. All the tasks, which do not violate the throughput, i.e., the hyperperiod $r_{out} \cdot T_{out}$, when assigned to EE processors are classified as EE tasks, Line 3, and all the rest as PE tasks, Line 4. In this way we guarantee that the throughput requirement will be met while minimizing the energy consumption.

Since the sorting algorithm in Line 1 has the worst-case complexity of $O(|V| \log |V|)$ and the worst-case complexity of the binary search in Line 2 is $O(\log |V|)$, the worst-case complexity of Algorithm 2 is $O(|V| \log |V|)$.

5.3 Task Mapping for Energy Minimization

In Algorithm 1, once the actors in a graph are classified by Algorithm 2 in Lines 5 and 15 into two sets of EE and PE actors, each set is mapped by Algorithm 3 in Lines 6 and 16 onto the corresponding type of clusters, EE and PE clusters, such that the energy consumption of the whole cluster heterogeneous MPSoC is minimized. Our algorithm of energy-efficient tasks mapping is given in Algorithm 3.

Algorithm 3 takes sets V^{EE} and V^{PE} of actors and a cluster heterogeneous MPSoC, and it returns the task mapping on processors in the clusters C, a vector of operating frequencies for clusters $\vec{F}$ and the minimum energy consumption E. The authors in [1] showed that the most balanced workload distribution leads to the least energy consumption, and that the the most balanced distribution is obtained when the Worst-Fit Decreasing (WFD) heuristic [6] is used to allocate tasks to processors. Thus, in this work, we use the WFD heuristic for task allocation. First, Algorithm 3 checks in Lines 1 to 4 whether the input MPSoC has enough resource to map (allocate) and schedule the tasks by using the WFD allocation heuristic [6], applied among the processors of the same type, and a given per-processor schedulability test [17] when processors are running at the maximum available frequency for each processor type. If there is no enough EE type of processors, we select some actors from set V^{EE} and assign them to set V^{PE}. The actors are selected in order of decreasing utilization and the selection is terminated as soon as the tasks corresponding to actors 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 the failure by returning an empty set for tasks-to-processors mapping C in Line 4. Line 5 checks if there are tasks that should be mapped on processors in EE clusters. If no task should be mapped to EE clusters, then EE clusters will not be used within the input MPSoC, hence they will not contribute to the total energy consumption, Line 6. Otherwise, the bounds on the number of active EE clusters are calculated in Line 8 and the energy consumption of mapping task set V^{EE} to EE clusters is calculated in Line 9. The lower bound n_{lb}^{EE} corresponds to the minimum possible number of active clusters to schedule the tasks because it is determined according to the ceiling of the utilization u^{EE} of EE tasks. The upper bound n_{ub}^{EE} is selected to be the minimum value among the case when tasks are mapped onto processors in one-to-one manner, and the case when all clusters available on the platform are active. We find the minimum energy for mapping the tasks on EE clusters by using Algorithm 4 (described later) in Line 9. Similarly, Line 10 checks whether there are tasks that should be mapped onto processors in PE clusters. If there are such tasks, lower and upper bounds of active PE clusters are calculated in Line 13 and the minimum energy for mapping the tasks on PE clusters by using Algorithm 4 is obtained in Line 14. Finally, the EE solution and the PE solution mappings are grouped together in Line 15 and the integral solution mapping of the given tasks onto the given MPSoC which results in minimum energy consumption is returned in Line 16 of Algorithm 3.

Within Algorithm 3, described above, Algorithm 4 is used to map the tasks which are in the same group, EE or PE, such that the energy consumption is minimized. Algorithm 4 takes the bounds on the number of active clusters of certain type (PE or EE), n_{lb} and n_{ub}, tasks V that are going to be mapped onto PE/EE clusters, the corresponding equation, Eq. (11) or (12) – see Section 4.5, for the calculation of the energy consumption and returns the task partitions among the processors in the clusters C_{best} and a vector of operating frequencies for clusters $\vec{F}_{best}$ which lead to minimal energy consumption E_{best}. In Lines 2 to 15 in Algorithm 4, the best task mapping and the frequency assignment is determined among different number of active clusters in the range from n_{lb} to n_{ub}. For each number of active clusters n, $n \in [n_{lb}, n_{ub}]$, the algorithm in Line 4 performs the WFD allocation heuristic [6] and uses a given per-processor schedulability test [17] to check the schedulability of the tasks. In this way, we want to achieve load balancing among the processors of the same type. If all tasks are allocated on processors, Line 5, we group processors into clusters according to their workload such that all processors in one cluster run at the frequency which matches their workload as much as possible. This is done in Lines 6 and 7, where processors $\pi_j \in \Pi$ are first sorted in non-increasing order of their workload, i.e., their utilization u_j, and then starting from the processor with the highest

Algorithm 4: Procedure to find the minimum energy when the given tasks are mapped onto the same type of clusters.

Input: Lower n_{lb} and upper n_{ub} bound on the number of clusters, set V of tasks that should be mapped onto clusters, equation $Eq.$ for calculating the energy consumption.
Output: Task mapping to processor in the clusters C_{best}, vector of operating frequencies for clusters $\vec{F}_{best}$ and the minimum energy consumption E_{best}.

1 $E_{best} = \infty$, $\vec{F}_{best} \leftarrow \emptyset$, $C_{best} \leftarrow \emptyset$;
2 **for** $n = n_{lb}$ *to* n_{ub} **do**
3 Create a set Π of $n \times N_p$ empty processors, $\forall \pi_j \in \Pi : u_j = 0$;
4 Perform WFD allocation heuristic and a corresponding schedulability test for all tasks in V;
5 **if** *all* $\tau_i \in V$ *can be scheduled on* Π **then**
6 $\Pi \leftarrow$ Sort Π in non-increasing order of u_j;
7 $C \leftarrow$ group every N_p processors in Π to a cluster C_k, $k \in [1, n]$;
8 $E = 0$, $F_k = 0, k \in [1, n]$;
9 **for** *cluster* $C_k \in C$ **do**
10 Find processor $\pi_j \in C_k$ with the highest utilization u_j, $u_{max} = u_j$;
11 Compute frequency of C_k as $F_k \geq u_{max} \cdot f_{max} \wedge F_k \in \mathcal{F}$;
12 Calculate energy E_k for cluster C_k by using $Eq.$;
13 $E = E + E_k$;
14 **if** $E < E_{best}$ **then**
15 $E_{best} = E$, $\vec{F}_{best} \leftarrow \vec{F}$, $C_{best} \leftarrow C$;
16 **return** E_{best}, $\vec{F}_{best}$, C_{best};

utilization, every N_p processors are grouped into a cluster. For each cluster, we select the smallest frequency which guarantees the schedulability and is supported by the cluster type, i.e., it is in the set $\mathcal{F}$ of available frequencies, Lines 9 to 11. The energy consumption of the mapping is calculated in Lines 12 and 13 of Algorithm 4. In Lines 14 and 15, we check whether the energy consumption obtained by mapping the tasks on the current number of active clusters n is the smallest one obtained so far. If that is the case, the mapping on the current number of active clusters becomes the best mapping solution. Finally, in Line 16, Algorithm 4 returns the minimum energy E_{best} obtained after mapping the tasks on clusters of the same type, the frequency assignment $\vec{F}_{best}$ for clusters and the cluster partitions C_{best}.

Let us now analyze the time complexity of Algorithm 4 and Algorithm 3 in the worst case. The complexity of Algorithm 4 is determined by the **for loop** in Lines 2 to 15. Due to the sorting algorithms used within the WFD heuristic, in Lines 4, and in Line 6, the complexity of Algorithm 4 is $O(N_c|V| \log |V|)$, where N_c is the number of active clusters. The worst-case complexity of Algorithm 3 is then determined by Line 2, which is executed in the worst case $|V|$ times, and every time the WFD allocation heuristic is applied, thus the complexity of Algorithm 3 is $O(|V|^2 \log |V|)$.

6. EVALUATION

We performed three experiments to evaluate the efficiency of our DPEM approach in comparison to the related energy minimization approaches in [7], [18], [25] and [15]. We selected the approaches in [7] and [18] for comparison because they consider the same task and system models as we do. We selected to compare with the approach in [25] because it is a very good representative among the approaches for energy-efficient mapping and scheduling of streaming applications modeled as SDF graphs. Finally, we compare our approach with the approach in [15] which is the only approach among the related approaches which consider task replication for energy minimization for classical periodic real-time tasks. In the first two experiments, we compare the approaches when the streaming applications are executed on a cluster heterogeneous platform. We apply our task classification method, given in Algorithm 2, for the approaches in [25] and [15] which were originally devised for homogeneous platforms and then we apply these approaches on the two sets of tasks, PE and EE, obtained by the classification. Since two of the related approaches, [25] and [15], originally consider homogeneous platforms with per-core VFS capability, in the third experiment, we compare our approach with these related approaches on this type of platform.

Table 2: Benchmarks used for evaluation.

Application	$\|V\|$	$\|E\|$	$\mathcal{R}$[1/time unit]
Discrete cosine transform (DCT)	8	7	1/47616
Fast Fourier transform (FFT)	17	16	1/12032
Filterbank	85	99	1/11312
Time delay equalization (TDE)	29	28	1/36960
Data encryption standard (DES)	53	60	1/1024
Serpent	120	128	1/3336
Bitonic Sorting	40	46	1/95
MPEG2	23	26	1/7680
Vocoder	114	147	1/9105
FMRadio	43	53	1/1434
Channel Vocoder	55	70	1/35500

The experiments were performed on the real-life applications from the StreamIt benchmarks suit [28], given in Table 2. $|V|$ denotes the number of actors in an SDF graph, while $|E|$ denotes the number of communication channels. $\mathcal{R}$ is the maximum achievable throughput, computed by using Eq. (3), (4) and (7), when the applications are scheduled by the ISPS approach described in Section 4.3. We consider these throughput values as the throughput constraints in our experiments.

In the experiments on heterogeneous MPSoC platforms, we consider the same MPSoC platforms considered in [18]. Those platforms have the same number of PE processors and EE processors but they have different cluster granularities, i.e., different number of processors per cluster, and hence, different number of clusters. We use the same MPSoC notation MPSoC_x_pe_ee as in [18]. For example, MPSoC_2_20_28 corresponds to an MPSoC platform with 2 processors per cluster, 20 PE clusters and 28 EE clusters. The approaches in [7], [18] and [15] use hard real-time scheduling algorithms to schedule the tasks on an MPSoC while the approach in [25] uses self-timed scheduling. The application tasks are permanently assigned to processors in [7], [15] and [25], while in [18], the tasks are permanently assigned to clusters, but within a cluster tasks are scheduled by a global scheduling algorithm, hence, they can migrate. In the experiments, we use the EDF [17] scheduling algorithm within our DPEM approach which is also used in [7] and [15]. In all experiments, we use the power parameters in [18] obtained from real measurements performed on the ODROID XU-3 [22] board. The results of the evaluations are shown in Figure 3, Figure 4 and Figure 5. In all these figures, we show the energy reduction obtained by our DPEM approach in comparison with the related approaches. The energy reduction r is computed by:

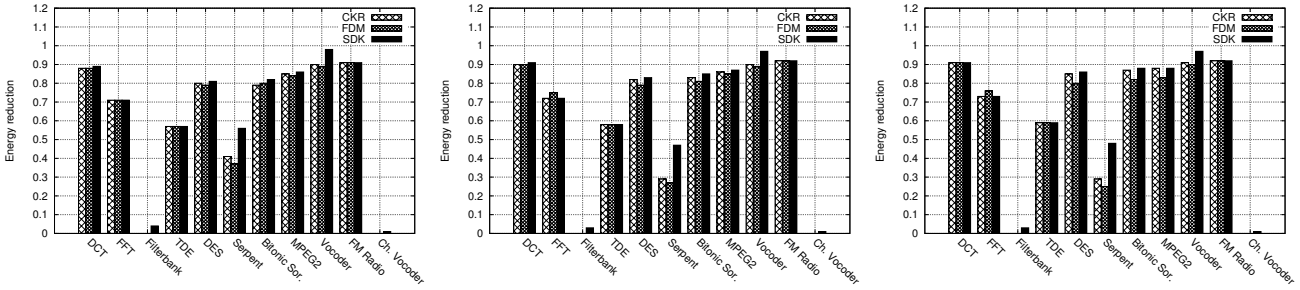
$$r = \frac{E_{rel} - E_{DPEM}}{E_{rel}}, \tag{13}$$

where E_{rel} is the energy consumption of an application to MPSoC mapping configuration obtained by a related approach and E_{DPEM} denotes the energy consumption achieved by our DPEM approach.

6.1 Comparison with [7], [18], [25] on Heterogeneous MPSoCs

In this section, we compare the energy consumption on cluster heterogeneous MPSoCs obtained by our proposed DPEM approach with the energy consumption delivered by the related approaches which do not consider task replication [7] – CKR, [18] – FDM, [25] – SDK.

The comparison results with the CKR, FDM and SDK approaches on the three considered heterogeneous MPSoCs are given in Figure 3(a)-3(c). In each of these figures, the x-axis shows the application benchmarks and the y-axis shows the energy reduction. Both approaches CKR and FDM are devised for cluster heterogeneous MPSoCs and both of them use preemptive hard real-time scheduling algorithms, which is also the case in our DPEM algorithm. We can see in Figure 3 that our DPEM approach reduces



(a) MPSoC_2_20_28 (higher is better) (b) MPSoC_4_10_14 (higher is better) (c) MPSoC_8_5_7 (higher is better)

Figure 3: Comparison of our proposed DPEM approach with related approaches on heterogeneous MPSoCs.

the energy consumption when compared to CKR and FDM for all but two considered benchmarks. The two benchmarks for which our approach results in the same energy consumption as CKR and FDM are Filterbank and Channel Vocoder. The workload which these two benchmarks put on the considered MPSoCs is balanced among processors, hence our approach will not replicate tasks of these benchmarks which leads to the same energy consumption as obtained by the CKR and FDM approaches. The average energy reduction of our approach when compared to the CKR approach is 62%, 62% and 63.1% for the three MPSoCs with 2, 4 and 8 processors per cluster, respectively. When compared to the FDM approach the corresponding average energy reductions are 61.6%, 61.4% and 61.6%. When compared to the SDK approach, our approach achieves energy reduction for all benchmarks because we use both task replication and preemptive scheduling. Note that we only use the design-time phase in the SDK approach for the comparison because our approach is a design-time approach. Our approach obtains on average the energy reduction of 65%, 65.1% and 66% for the three MPSoCs with 2, 4 and 8 processors per cluster, respectively, when compared to the SDK approach. We can conclude from these results that our approach achieves big energy reduction by utilizing task replication.

6.2 Comparison with [15] on Heterogeneous MPSoCs

In this section, we compare the energy consumption on cluster heterogeneous MPSoCs of our DPEM approach with the related approach in [15], denoted by WYL, which considers task replication as well. The results are given in Figure 4. Here again, both approaches will not replicate tasks in Filterbank and Channel Vocoder and hence both approaches will lead to the same energy consumption in these two cases. Given that the task classification in the WYL approach is based on the power consumption curve of a processor, the WYL approach will never replicate tasks assigned to EE processors. In addition, the WYL approach will never replicate the tasks of an application once the total number of heavy tasks is equal to the number of processors on an MPSoC platform. All these limitations of WYL explain the energy reduction achieved when our approach is used to map the benchmarks in Table 2 onto the three considered MPSoCs. The average energy reduction obtained by our DPEM approach is 51.3%, 57.2% and 60.7% for the MPSoCs with 2, 4 and 8 processors per cluster, respectively.

6.3 Comparison on Homogeneous MPSoC

Given that both the SDK and WYL approaches were originally proposed for homogeneous platforms with per-core VFS capability, in this section, we compare the energy consumption on such systems when our DPEM approach is used with the energy consumption values when the SDK and WYL approaches are used. The results of the energy reduction on a homogeneous MPSoC platform consisting of 96 PE processors with per-core VFS capability are given in Figure 5. Here, we also give the results of energy reduction when our DPEM approach is compared with the CKR and FDM approaches for completeness.

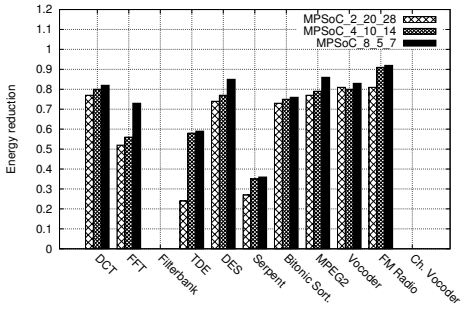


Figure 4: Comparison between DPEM and WYL on heterogeneous MPSoCs.

The benchmarks Filterbank and Channel Vocoder were the only two benchmarks for which our approach could not obtain any reduction in energy consumption on heterogeneous MPSoCs when compared to the approaches which use hard real-time scheduling algorithms – CKR, FDM and WYL. In the case of a homogeneous platform, we can see in Figure 5 that there is still no difference in energy consumption between our DPEM and the CKR, FDM and WYL for the Filterbank benchmark. This happens because when mapped onto a homogeneous MPSoC, Filterbank has balanced workload among the processors, hence both our DPEM and the WYL approaches will not replicate tasks. However, in the case of the Channel Vocoder benchmark, we see in Figure 5 that the situation changes, i.e., now there is a reduction in energy when our approach is compared to the CKR and FDM, because our approach will replicate tasks to balance the workload of Channel Vocoder on a homogeneous platform. The WYL approach will replicate tasks as well, leading to the same energy consumption as obtained by our DPEM approach. Although the WYL approach was devised for homogeneous platforms with types of processors which match the PE type and with per-core VFS capability, still our DPEM approach outperforms the WYL approach by reducing energy on average by 10.4%, and in the best case up to 22%. The reason is that our task replication procedure is more flexible than the procedure in the WYL approach.

When compared to the another approach devised for homogeneous MPSoCs with per-core VFS capability, i.e., the SDK approach, our DPEM approach leads to an energy reduction of 36% on average and up to 90% in the best case. The reason is that our approach replicates tasks to lower the utilization per-processor, and hence, lower operating frequencies can be achieved. In addition, the SDK approach minimizes only the dynamic energy consumption and uses non-preemptive scheduling which both lead to higher total energy consumption.

Finally, when compared to the CKR and FDM approaches on a homogeneous platform, our DPEM approach delivers systems with energy reduction of 21.2% and 25.6% on average, respectively. The main reason is the task replication which our approach uses

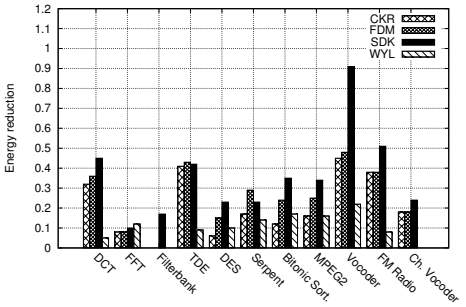


Figure 5: Comparison on homogeneous MPSoC.

to lower the utilization per processor while keeping the application throughput.

We performed an additional experiment to evaluate the influence of the number of processors in an MPSoC on the energy reduction of our DPEM approach in comparison with the related approaches. In this experiment, beside the MPSoC platform with 96 PE processors, we considered two additional platforms with 48 and 192 PE processors with per-core VFS capability. On the 48-processor platform, our DPEM approach resulted in the energy reduction of 6%, 18.5%, 20.5% and 30.3% when compared with the WYL, CKR, FDM and SDK approaches, respectively. In comparison with the same related approaches, our DPEM approach obtains on the 192-processor platform the following energy reductions – 10.7%, 24.9%, 29.4% and 39.8%. We can conclude that the energy reduction of our approach with regard to the related approaches slowly increases with the increase of the number of processors in the platform.

6.4 Overhead and time complexity analysis

In this section, we briefly discuss the code and data memory overhead of our approach when compared to the related approaches and the time complexity of our and the related approaches. The code and data memory overhead of our approach on heterogeneous platforms when compared to the WYL approach is 2 times higher on average, and 2.3 times higher on average than the approaches which do not consider task replication, i.e., approaches CKR, FDM and SDK. The memory overhead of our DPEM approach on the homogeneous platform is 16% higher on average when compared to the WYL approach, and 85% higher on average when compared to the CKR, FDM and SDK approaches. Given that the actual memory increase in the worst case is 213 KB and given the size of memory available in modern embedded systems, we can conclude that the memory overhead introduced by our approach is acceptable.

The time complexity in the worst-case of our DPEM approach and the approaches CKR, FDM and WYL is polynomial, while the worst-case time complexity of the SDK approach is exponential. In the worst-case, our approach needs 62 minutes, the WYL approach needs 5 minutes, the CKR approach takes 11 minutes, the FDM less than 1 second and the SDK approach needs 6 days to find an energy-efficient solution. Given that our DPEM approach is a design-time approach and that it delivers solutions of better quality, we can conclude that our approach outperforms the related approaches.

7. CONCLUSIONS

In this paper, we presented a novel polynomial-time energy minimization mapping approach for SDF graphs which uses task replication to achieve load-balancing on processors of the same type, which enables processors to run at a lower frequency, consuming less energy. The experiments on a set of real-life streaming applications showed that our approach reduces energy consumption by 66% on average while meeting the same throughput requirement when compared to related energy minimization mapping approaches.

8. REFERENCES

[1] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *IPDPS*, 2003.

[2] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1):7:1–7:34, 2016.

[3] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers. In *RTSS*, 2010.

[4] G. Bilsen et al. Cyclo-static dataflow. *IEEE Trans. Signal Process.*, 44(2):397–408, 1996.

[5] D. Bui and E. A. Lee. Streamorph: A case for synthesizing energy-efficient adaptive programs using high-level abstractions. In *EMSOFT*, 2013.

[6] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. 1996.

[7] A. Colin, A. Kandhalu, and R. Rajkumar. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *RTSCA*, 2014.

[8] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Throughput-constrained dvfs for scenario-aware dataflow graphs. In *RTAS*, 2013.

[9] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35, 2011.

[10] P. Greenhalgh. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7, 2011.

[11] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED*, 2007.

[12] S. Holmbacka, E. Nogues, M. Pelcat, S. Lafond, D. Menard, and J. Lilius. Energy-awareness and performance management with parallel dataflow applications. *J. of Signal Processing Systems*, pages 1–16, 2015.

[13] P. Huang, O. Moreira, K. Goossens, and A. Molnos. Throughput-constrained voltage and frequency scaling for real-time heterogeneous multiprocessors. In *SAC*, 2013.

[14] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[15] W. Y. Lee. Energy-saving dvfs scheduling of multiple periodic real-time tasks on multi-core processors. In *DS-RT*, 2009.

[16] D. Li and J. Wu. Energy-aware scheduling for acyclic synchronous data flows on multiprocessors. *J. of Interconnection Networks*, 14(4), 2013.

[17] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[18] D. Liu, J. Spasic, G. Chen, and T. Stefanov. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *ESTIMedia*, 2015.

[19] T. Mitra. Heterogeneous multi-core architectures. *IPSJ Trans. on System LSI Design Methodology*, 8:51–62, 2015.

[20] A. Nelson, O. Moreira, A. Molnos, S. Stuijk, B. T. Nguyen, and K. Goossens. Power minimisation for real-time dataflow applications. In *DSD*, 2011.

[21] nVidia. NVIDIA Tegra X1: NVIDIA'S New Mobile Superchip, 2015.

[22] ODROID. http://www.hardkernel.com.

[23] M. Sackmann, P. Ebraert, and D. Janssens. A fast heuristic for scheduling parallel software with respect to energy and timing constraints. In *IPDPSW*, 2011.

[24] Samsung. http://www.samsung.com.

[25] A. K. Singh, A. Das, and A. Kumar. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *DAC*, 2013.

[26] J. Spasic, D. Liu, E. Cannella, and T. Stefanov. Improved hard real-time scheduling of csdf-modeled streaming applications. In *CODES+ISSS*, 2015.

[27] J. Spasic, D. Liu, and T. Stefanov. Exploiting resource-constrained parallelism in hard real-time streaming applications. In *DATE*, 2016.

[28] W. Thies and S. Amarasinghe. An empirical characterization of stream programs and its implications for language and compiler design. In *PACT*, 2010.

[29] Y.-H. Wei, C.-Y. Yang, T.-W. Kuo, S.-H. Hung, and Y.-H. Chu. Energy-efficient real-time scheduling of multimedia tasks on multi-core processors. In *SAC*, 2010.

[30] H. Xu, F. Kong, and Q. Deng. Energy minimizing for parallel real-time tasks based on level-packing. In *RTCSA*, 2012.

[31] J. Zhu, I. Sander, and A. Jantsch. Energy efficient streaming applications with guaranteed throughput on mpsocs. In *EMSOFT*, 2008.