Energy and Feasibility Optimal Global Scheduling Framework on big.LITTLE platforms

Hoon Sung Chwa^{*}, Jaebaek Seo^{*}, Hyuck Yoo^{*} Jinkyu Lee[†], Insik Shin^{*} ^{*}Department of Computer Science, KAIST, Republic of Korea [†]Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea insik.shin@cs.kaist.ac.kr

Abstract—Motivated by ARM's big.LITTLE, the cutting-edge heterogeneous multi-core architecture that supports migration between cores with different performance and energy efficiency, this paper targets global heterogeneous multi-core scheduling, and achieves the optimality in terms of energy consumption and feasibility. To this end, we address the problem of determining not only the system static configurations such as on-and-off status and voltage/frequency, and but also the time-varying schedule of each task. First, we abstract each task's schedule as a rate of each task's workload on each cluster (big and LITTLE) and formulate an optimization problem that achieves both energy and feasibility optimality. We then develop a time-efficient task workload allocation algorithm to assign the workload rate of each task and the system configurations. To generate task schedules from the algorithm, we establish feasibility-optimal scheduling rules for a two-type heterogeneous multi-core platform, which generalize the existing rules for a homogeneous one. Our simulation results demonstrate that our approach yields up to 37% less energy consumption without compromising feasibility than best-possible partitioned scheduling approaches obtained by solving an ILP optimization formulation.

I. INTRODUCTION

Advances on chip architectures towards multi-cores have provided two general options for real-time scheduling: global scheduling that allows a task to migrate from one core to another, and partitioned scheduling that disallows the migration. To explore their own advantages such as small overhead of the former and full utilization of the latter, real-time multicore scheduling has been widely studied, mostly for homogeneous cores [1]. When it comes to heterogeneous multi-core scheduling, very few studies have focused on global scheduling mainly due to insufficient architectural support for migration in commercial heterogeneous multi-core chips.

Recently, ARM has launched a two-type heterogeneous multi-core chip, called big.LITTLE [2], which has been deployed in the state-of-the-art smartphones, e.g., Samsung Galaxy 4 and Note 3. The big.LITTLE architecture consists of two types of cores: one with high-performance "big" cores and the other with power-efficient "LITTLE" cores. One of the most distinguishable features of the big.LITTLE architecture is a practical support for migration; cores in two different types (big and LITTLE) not only deploy the same instruction-set architecture, but also share a specially designed interconnection bus for data transfer between the clusters. Through coupling big and LITTLE cores, the big.LITTLE architecture is capable of global scheduling to achieve highperformance with maximum energy efficiency. In real-time systems research, however, almost all energy-aware scheduling approaches for heterogeneous multi-core platforms have been focused on partitioned scheduling rather global scheduling [3], [4], [5], [6].

Motivated by the cutting-edge heterogeneous multi-core architecture, we focus on global scheduling, and demonstrate "how good global scheduling is for a big.LITTLE platform" compared to existing partitioned scheduling approaches from the both core utilization and energy consumption points of view. To this end, we would like to achieve the following goals: (a) feasibility optimality—our solution can schedule all jobs in a task set without any job deadline miss, if there exists such a feasible solution, and (b) energy-optimality—any feasible solution cannot result in less energy consumption than our solution. Then, we need to determine the following cluster configurations and job schedules to achieve the goals: (i) the onand-off status of each cluster, (ii) the static voltage/frequency level of each cluster, and (iii) schedule of each job (i.e., when and where each job executes).

Since it is too complicated to determine (i)-(iii) at once, especially (iii) during the entire time interval of interest, we abstract each task's schedule as a rate of each task's workload on each cluster. Then, we divide the problem into two: (1) determining the cluster-level static configurations and the workload rate of each task on each cluster, and (2) developing a global scheduling algorithm that generates task schedules from the input assigned in (1). For (1), we reduce search space by deriving necessary feasibility conditions, and then develop a task workload allocation algorithm that achieves both energy and feasibility optimality. When it comes to (2), we establish feasibility-optimal scheduling rules for a two-type heterogeneous multi-core platform, which generalize existing rules for a homogeneous one (called DP-Fair [7]).

To evaluate our approach, we perform simulations with real system parameters from a big.LITTLE core. Our simulation results demonstrate that the proposed energy/feasibilityoptimal global scheduling framework yields up to 37% less energy consumption without compromising feasibility than best-possible partitioned scheduling configurations obtained by solving an ILP optimization formulation.

In summary, this paper offers the following contributions.

- C1 Addressing a need for global heterogeneous scheduling, motivated by the cutting-edge architecture, called big.LITTLE;
- C2 Development of the first energy-aware global scheduling framework for a big.LITTLE platform consisting of C3 and C4;
- C3 Development of a low time-complexity optimal task

workload allocation algorithm for big.LITTLE based on necessary feasibility conditions we derived; and

- C4 Establishment of feasibility-optimal scheduling rules for a two-type heterogeneous multi-core platform, which generalizes DP-Fair rules for a homogeneous one; and
- C5 Demonstration of the effectiveness of our solution via simulation.

The rest of the paper is structured as follows. Section II presents our system model and model validation, followed by the problem statement in Section III. Section IV formulates an optimization problem that determines system configurations and a ratio of workload of each task on the big and LIT-TLE clusters, and develops a time efficient energy/feasibility-optimal task workload allocation algorithm. Section V provides feasibility-optimal scheduling rules for implicit-deadline periodic tasks running on two-type heterogeneous multi-core platforms. Section VI evaluates our energy/feasibility global scheduling framework. Section VII discusses related work, and finally Section VIII concludes this paper.

II. SYSTEM MODEL AND VALIDATION

In this section, we present our system model including a big.LITTLE platform, task model, and power model. Then, we validate our system model via experiments on a real big.LITTLE processor.

A. System model

big.LITTLE platforms. The big.LITTLE architecture is a computing platform consisting of two heterogeneous clusters: one with high-performance big cores and the other with power-efficient LITTLE cores. Due to the nature of the big.LITTLE architecture, a big core exhibits high energy consumption with high performance, while a LITTLE one does the opposite behaviors. Both clusters share not only the same instruction-set architecture (ISA), but also a specially designed interconnection bus for data transfer between the clusters [2]. Therefore, it is practical for a task in one cluster to migrate to the other cluster *in the middle of execution*, which cannot be realized in most existing heterogeneous multi-core architectures.

The big.LITTLE architecture provides dynamic voltage and frequency scaling (DVFS) per cluster. The big (likewise LITTLE) cluster provides nine (likewise five) discrete frequency/voltage levels as shown in Table I [8]. We note that the big.LITTLE architecture supports only cluster-level DVFS, meaning that we can apply different voltage/frequency to the big and LITTLE clusters, but cores in the same cluster operate with the same frequency/voltage. Let f_B (V_B) and f_L (V_L) denote the frequency (voltage) of a core in the big and LITTLE cluster, respectively. Among the several frequency options in Table I, let $f_{B,max}$ and $f_{L,max}$ denote the maximum frequency for a core in the big and LITTLE cluster, respectively.

We denote the number of cores in the big and LITTLE cluster is m_B and m_L , respectively.

Task model. We consider an implicit-deadline periodic task model, in which a task τ_i in a task set τ is characterized by (T_i, C_i^B, C_i^L) : the period or the relative deadline T_i , and the worst-case execution time (WCET) at the maximum frequency

upon the big and LITTLE core (denoted by C_i^B and C_i^L , respectively). In general, the WCET is in inverse proportional to frequency; therefore, if a big core operates with a given frequency f_B , the WCET of τ_i is calculated by $C_i^B \cdot \frac{f_{B,max}}{f_B}$. Likewise, the WCET of τ_i on a LITTLE core with a given frequency f_L is $C_i^L \cdot \frac{f_{L,max}}{f_L}$, to be validated in Section II-B.

Each task τ_i generates a potentially infinite sequence of jobs for every T_i time units, and each job released by a task τ_i has to complete its execution within T_i time units from its release. We assume that jobs are independent, i.e., they do not share any resources except cores and do not have any data dependencies. A single job cannot be executed upon more than one core (regardless of core type) in parallel. As supported by the big.LITTLE architecture, a job can migrate from a big core to a LITTLE one (or from a LITTLE core to a big one); in this case, the amount of execution of τ_i performed on a LITTLE core corresponds to that on a big core multiplied by C_i^L/C_i^B . For example, suppose that $C_i^B = 4$ and $C_i^L = 8$; both cluster operate with the maximum frequency; and a job of τ_i executes one time unit on a big core and then migrates to a LITTLE core. In this case, one time unit execution on a big core corresponds $1 \cdot C_i^L / C_i^B = 2$ time units execution on a LITTLE core. Therefore, after migration, the job has 8-2=6time units execution left on a LITTLE core.

Power model. In a big.LITTLE architecture, there are three power states of a core: *off, idle,* and *active.* Under the off state, a core is turned off, and it cannot execute any job until the core becomes activated (note that it takes time and power to activate). Once a core is activated (i.e., turned on), the core is on the either idle or active state. A core is said to be on the active state if the core has a currently-executing job, or on the idle state otherwise. Then, the power consumption of a core (denoted by $P_{\rm core}$) is expressed as

$$P_{\rm core} = P_{\rm static} + P_{\rm dynamic},\tag{1}$$

where P_{static} is the power for the core to keep ready to execute (on the either active or idle state), and P_{dynamic} is the additional power to execute a job. In other words, a core on the off, idle, and active state exhibits (a) $P_{\text{static}} = P_{\text{dynamic}} = 0$, (b) $P_{\text{static}} > 0$ and $P_{\text{dynamic}} = 0$, and (c) $P_{\text{static}}, P_{\text{dynamic}} >$ 0, respectively. The two terms are modelled as follows [9]: $P_{\text{static}} = C_{\text{static}} \cdot V^{\rho}$ and $P_{\text{dynamic}} = C_{\text{dynamic}} \cdot fV^2$, where C_{static}, ρ , and C_{dynamic} are constants depending on core types.

In addition to the power consumption of each core, a cluster of a big.LITTLE processor consumes power so as to support cores in the cluster to execute, called the uncore power consumption (denoted by P_{uncore}) [9]. That is, if at least one core in a cluster is on the either idle or active state, P_{uncore} is positive; otherwise, P_{uncore} is zero.

In order to obtain the values of the hardware dependent parameters (i.e., $C_{\rm static}$, ρ , and $C_{\rm dynamic}$), we measured the power consumption on a big.LITTLE development board for available operating frequency/voltage combinations, to be presented in Section II-B.

B. Model validation

Experiment setup. We use the ODROID-XU+E board [10] comprising of four Cortex-A15 ("big") cores along with four

big									LITTLE						
v_B : voltage (V)	0.93	0.96	1.0	1.04	1.08	1.1	1.15	1.2	1.23	v_L : voltage (V)	0.9	0.94	1.01	1.09	1.2
f_B : frequency (MHz)	800	900	1000	1100	1200	1300	1400	1500	1600	f_L : frequency (MHz)	250	300	400	500	600

TABLE I. FREQUENCY/VOLTAGE LEVELS OF A BIG AND LITTLE CORE



Fig. 1. Model validation

Cortex-A7 ("LITTLE") cores. The ODROID-XU+E board is equipped with sensors to measure the power consumption of the big and LITTLE clusters individually. In our experiments, we utilize only one big and one LITTLE core. The remaining cores in each cluster are logically turned off using system calls, such that no jobs are scheduled on them. We set the voltage and frequency for each cluster using the Linux userspace governor.

We use five types of benchmarks: CPU intensive, cache intensive, memory intensive, I/O intensive with the buffer cache, and I/O intensive without the buffer cache. In the CPU intensive benchmark, a process runs a busy loop with no memory accesses. In the cache intensive benchmark, a process strides through a memory region performing readmodify-write cycles on successive cache lines. The size of the region is twice of the L1 cache size. A process in the memory intensive benchmark is the same as that in the cache intensive benchmark except increased working set size to twice of the L2 cache size of the big cluster. In the I/O intensive benchmarks, a process writes an image to a file with/without a buffer cache.

Task execution model validation. In order to validate our task execution model, we run the above five types of benchmarks 25 times for each frequency/voltage level, and measure the average CPU time. Figure 1(a) shows that each benchmark exhibits different performance between the big and LITTLE clusters, but the CPU time is in inverse ratio to frequency levels within the same cluster. Thereby, the formular

	A15 core (big)	A7 core (LITTLE)
C_{static}	1.478	1.191
ρ	0.379	0.757
$C_{\rm dynamic}$	0.471	0.153

 TABLE II.
 POWER MODEL PARAMETERS FOR A BIG AND LITTLE CORE

 $C_i^B \cdot \frac{f_{B,max}}{f_B}$ discussed in our task model is valid in calculating the WCET of τ_i on a big core with a given frequency f_B . Likewise, $C_i^L \cdot \frac{f_{L,max}}{f_L}$ is valid on a LITTLE core.

Power model validation. To validate that our power model adequately represents real hardware behaviors, we measure the real-time power consumption and obtain the model parameters. We run the CPU intensive benchmark for each frequency level during 5 minutes, and read the power sensor data. The measurement results are shown in Figure 1(b). We choose a linear regression method to obtain the parameters, and Table II shows the estimated parameters.

III. PROBLEM STATEMENT

As mentioned in the introduction, this paper considers global preemptive scheduling on a big.LITTLE platform, and aims at achieving the following goals:

- G1. *Feasibility-optimality*—our solution schedules all jobs in a task set without any deadline miss of a job, as long as there exists such a feasible solution; and
- G2. *Energy-optimality*—any feasible solution cannot yield less energy consumption than our solution.

To achieve the goals, we determine the following cluster configurations and job schedules:

- D1. On-and-off status of each cluster;
- D2. Static voltage/frequency level of each cluster; and
- D3. Schedule of each job, i.e., the time intervals in and the core on which each job executes.

For D1, we have three options: (a) both big and LITTLE clusters are turned on; (b) only the big cluster is turned on; and (c) only the LITTLE cluster is turned on. Note that we do not consider a core-level on-and-off policy, because it has been demonstrated that P_{static} is negligible compared to P_{uncore} [9], implying we hardly benefit from the core-level on-and-off policy in terms of energy consumption. Thereby, we assume that all cores are activated in a cluster when the cluster is turned on. Let δ_B and δ_L denote the on-and-off status of the big and LITTLE clusters, i.e., 1 if on, 0 otherwise.

When it comes to D2, we have cluster-level discrete choices as shown in Table I, since the big.LITTLE architecture does not support core-level voltage/frequency regulation. Here, we consider static (rather than dynamic) voltage/frequency scaling on which the operating voltage/frequency does not change over time. In the previous literatures [11], [12], the energy optimal frequency is a constant when each job presents its worstcase workload behavior with the convex power consumption function. Therefore, the static scaling not only is simpler, but also minimizes the worst-case energy consumption (when each job presents its worst-case execution time). Such a worstcase behavior of energy consumption is important for mobile, battery-powered devices in which a big.LITTLE processor is deployed.

For D3, it is too complex to determine all the job schedules during the entire time interval of interest, because we should decide not only "when", but also "where" to execute a job, which yields different speed of execution and energy consumption. Therefore, we abstract the job schedule as the ratio of workload of its invoking task. This is because, the ratio of workload of a task not only indicates the amount of execution of the task on each cluster, but also determines the duration for a core to be on the active state due to the task's execution, which can be translated into the energy consumption. Let x_i^B and x_i^L denote the fraction ratio of workload for which a task τ_i executes on the big and LITTLE clusters, respectively, where $x_i^B + x_i^L = 1$. If $x_i^B = 1$ $(x_i^L = 1)$, τ_i is executed only on a big (LITTLE) cluster. If $0 < x_i^B, x_i^L < 1$ holds, τ_i is fractionally executed on both big and LITTLE clusters. Using the ratio of workload of each task on each cluster, we divide the problem into the following two steps. At step 1, we determine how much portion of a task workload will be executed on big and LITTLE clusters (i.e., x_i^B and x_i^L for every $\tau_i \in \tau$). Then, at step 2, we develop a global scheduling algorithm that generates job schedules for given x_i^B and x_i^L for every $\tau_i \in \tau$, determined by step 1.

We now summarize our approach to determining D1, D2 and D3 that satisfies G1 and G2 as follows.

Given a feasible task set τ of periodic real-time tasks and a big.LITTLE platform comprising m_B big cores and m_L LITTLE ones,

- Step 1. Determine the on-and-off status (δ_B and δ_L) and static frequency (f_B and f_L) of the big and LITTLE clusters and the ratio of workload of each task on the big and LITTLE clusters (x_i^B and x_i^L for every $\tau_i \in \tau$), such that they yield the energy-optimality without compromising feasibility; and
- Step 2. Develop a global scheduling algorithm that generates job schedules from the values assigned by step 1 such that all jobs meet their deadlines.

To summary, step 1 performs *energy/feasibility-optimal task workload allocation*, and step 2 develops *feasibility-optimal global scheduling algorithm*, which are presented in Sections IV and V, respectively. We then evaluate our solution in Section VI.

IV. ENERGY/FEASIBILTY-OPTIMAL TASK WORKLOAD ALLOCATION

In this section, we present our approach to determining the on-and-off status and static frequency of both clusters and the ratio of workload of each task on both cluster (i.e., δ_B , δ_L , f_B , f_L , $\{x_i^B\}$, and $\{x_i^L\}$), explained in step 1 in Section III. To this end, we first formulate an optimization problem that achieves both energy/feasibility optimality. Then, we derive necessary feasible conditions for the solution, and based on the conditions, we present a time-efficient task workload allocation algorithm.

A. Problem formulation

Based on our power model, we can calculate the energy consumption of each cluster in an interval. Since job schedules are repeated at every hyperperiod of a task set (denoted and calculated by $H = LCM(\cup_i T_i)$), we now calculate energy consumption in an interval of length H.

While P_{static} and P_{uncore} only depend on the on-andoff status of the big and LITTLE clusters (i.e., δ_B and δ_L), P_{dynamic} additional relies on the execution time of jobs in each cluster. To this end, we calculate the amount of execution of a job of τ_i in each cluster. Since the execution time of a job of τ_i is $C_i^B \cdot \frac{f_{B,max}}{f_B}$ if it is fully executed on a big core, $x_i^B \cdot C_i^B \cdot \frac{f_{B,max}}{f_B}$ is the amount of actual execution of a job of τ_i on a big core. Therefore, we calculate the utilization of τ_i 's execution on the big and LITTLE clusters (denoted by u_i^B and u_i^L) as follows:

$$u_i^B = x_i^B \cdot C_i^B \cdot \frac{f_{B,max}}{f_B} \cdot \frac{1}{T_i}, \quad u_i^L = x_i^L \cdot C_i^L \cdot \frac{f_{L,max}}{f_L} \cdot \frac{1}{T_i}.$$
 (2)

Note that u_i^B with $x_i^B = 1$ is said to be the maximum utilization of τ_i 's execution on the big cluster, and denoted by $u_i^{B,max}$; likewise, $u_i^{L,max}$ denotes u_i^L with $x_i^L = 1$.

Then, in an interval of length H, the cumulative execution time of jobs of τ_i on the big (LITTLE) cluster is $H \cdot u_i^B$ $(H \cdot u_i^L)$. Then, the cumulative energy consumption in the big and LITTLE clusters in an interval of length H (denoted by E_B and E_L) is calculated as follows:

$$E_{B} = \delta_{B} \cdot H \cdot \left(P_{\text{uncore}}^{B} + m_{B} \cdot P_{\text{static}}^{B} + P_{\text{dynamic}}^{B} \cdot \sum_{\tau_{i} \in \tau} u_{i}^{B} \right),$$

$$E_{L} = \delta_{L} \cdot H \cdot \left(P_{\text{uncore}}^{L} + m_{L} \cdot P_{\text{static}}^{L} + P_{\text{dynamic}}^{L} \cdot \sum_{\tau_{i} \in \tau} u_{i}^{L} \right).$$

$$(4)$$

Note that if the big (LITTLE) cluster is turned off, i.e., $\delta_B = 0$ ($\delta_L = 0$), the energy consumption of the big (LITTLE) cluster E_B (E_L) is zero. Also, for simplicity of presentation, we only concern "frequency" instead of "voltage/frequency"; as shown in Table I, if frequency is determined, the corresponding voltage is given.

Then, we formally present the optimization problem (denoted by Energy/Feasibility-OPT) of determining the on-andoff status and static frequency of both clusters and the ratio of workload of each task on both cluster (i.e., δ_B , δ_L , f_B , f_L , $\{x_i^B\}$, and $\{x_i^L\}$), as follows.

 $\begin{array}{ll} \text{Minimize} & E_B(\delta_B, f_B, \{x_i^B\}) + E_L(\delta_L, f_L, \{x_i^L\}),\\ \text{Subject to} & \text{C1: } \forall \tau_i \in \tau, x_i^B + x_i^L = 1,\\ & \text{C2: } \forall \tau_i \in \tau, u_i^B + u_i^L \leq 1,\\ & \text{C3: } \sum_{\tau_i \in \tau} u_i^B \leq m_B,\\ & \text{C4: } \sum_{\tau_i \in \tau} u_i^L \leq m_L,\\ & \text{C5: } \forall \tau_i \in \tau, 0 \leq x_i^B, x_i^L \leq 1. \end{array}$

Here, the objective function achieves the energy-optimality, while constraints C1–C5 yields feasibility-optimality. Constraint C1 specifies that every task must receive its appropriate amount of execution. Constraint C2 asserts that each task cannot be executed upon both clusters at the same time. Constraints C3 and C4 assert that total workload allocated on each cluster should be less than or equal to the capacity of each cluster. Those constraints correspond to the feasibility conditions presented in [13].

Note that if the big (LITTLE) cluster is turned off, we set m_B (m_L) to zero to apply non-availability of the cluster.

When we calculate the optimization problem, we do not assume any particular relation between the big and LITTLE cluster in terms of energy consumption and execution speed. In a real world, a big core consumes more power and takes less job execution time than a LITTLE core. However, we do not rely on such a relation when we calculate the optimization problem; instead, we seek to find a general solution.

B. Necessary feasible conditions

There are many ways to solve our optimization problem, but we would like to solve it in an efficient manner in terms of time-complexity. Since there are only several cluster configurations of the on-and-off status and static frequency, we make solutions of $\{x_i^B, x_i^L\}$ for all possible cluster configurations (i.e., δ_B , δ_L , f_B and f_L) and choose the best solution among all the solutions.

To solve our optimization problem for given cluster configurations, this subsection investigates some necessary conditions for each task's ratio of workload on the big and LITTLE clusters (i.e., $\{x_i^B, x_i^L\}$), which will be a basis for a solution algorithm to be presented in Section IV-C.

By the parallelism restriction constraint C2, if there exists a task τ_i such that $u_i^{B,max} > 1$ and $u_i^{L,max} > 1$ hold, the task cannot satisfy the constraint, which leads to infeasibility. However, if $u_i^{L,max} > 1$ and $u_i^{B,max} \leq 1$ hold, we may find a feasible solution by moving some workload of τ_i from the LITTLE to the big cluster. In this case, there must exist the minimum utilization of τ_i on the big core so as to satisfy the constraint C2. Conversely, if $u_i^{L,max} \leq 1$ and $u_i^{B,max} > 1$ hold, there must exist the minimum utilization of τ_i on the LITTLE core with the same reasoning. If $u_i^{L,max} \leq 1$ and $u_i^{B,max} \leq 1$ hold, a task τ_i always satisfies constraint C2. Recall that the maximum utilization of τ_i 's execution on the big and LITTLE clusters is $u_i^{B,max} = C_i^B \cdot \frac{f_{B,max}}{f_B} \cdot \frac{1}{T_i}$ and $u_i^{L,max} = C_i^L \cdot \frac{f_{L,max}}{f_L} \cdot \frac{1}{T_i}$, respectively. Then, the following lemma calculates such a minimum utilization.

Lemma 1: The minimum value of the utilization of each task on the big and LITTLE clusters (denoted by lo_i^B and lo_i^L) is calculated by

$$lo_{i}^{B} = \begin{cases} \frac{u_{i}^{L,max} - 1}{u_{i}^{L,max} - u_{i}^{B,max}}, & \text{if } u_{i}^{L,max} > 1, \\ 0, & \text{otherwise.} \end{cases}$$
(5)

$$lo_{i}^{L} = \begin{cases} \frac{u_{i}^{B,max} - 1}{u_{i}^{B,max} - u_{i}^{L,max}}, & \text{if } u_{i}^{B,max} > 1, \\ 0, & \text{otherwise.} \end{cases}$$
(6)

Proof: The minimum value of the utilization of each task on the big cluster (lo_i^B) is induced by constraint C2. In constraint C2, if we substitute x_i^L to $1-x_i^B$ based on constraint C1, we can calculate lo_i^B . The same holds for lo_i^L . Details are given in Appendix A.

Once lo_i^B , lo_i^L is calculated, the rest of workload ratio (i.e., $1 - lo_i^B - lo_i^L$ by constraint C1) for each task can be properly allocated to each cluster as long as cluster capacity constraints meet. We let y_i^B and y_i^L denote the workload ratio excluding lo_i^B and lo_i^L (i.e., $x_i^B = y_i^B + lo_i^B$ and $x_i^L = y_i^L + lo_i^L$), respectively. Then, the constraints C1–C5 for guaranteeing feasibility can be reduced as

$$\begin{split} &\overline{\text{C1}} \colon \forall \tau_i \in \tau, y_i^B + y_i^L = 1 - lo_i^L - lo_i^B, \\ &\overline{\text{C3}} \colon \sum_{\tau_i \in \tau} y_i^B \cdot u_i^{B,max} \leq m_B - \sum_i lo_i^B \cdot u_i^{B,max}, \\ &\overline{\text{C4}} \colon \sum_{\tau_i \in \tau} y_i^L \cdot u_i^{L,max} \leq m_L - \sum_i lo_i^L \cdot u_i^{L,max}, \\ &\overline{\text{C5}} \colon \forall \tau_i \in \tau, 0 \leq y_i^B, y_i^L \leq 1 - lo_i^L - lo_i^B. \end{split}$$

Note that constraint C2 is removed. This is because constraint C2 is never violated if x_i^B and x_i^L is assigned at least lo_i^B and lo_i^L by lemma 1, respectively.

C. Solution to the optimization problem

Since we reduce the problem by allocating lo_i^B and lo_i^L amount of utilization to the big and LITTLE clusters, respectively, the remaining step is to determine $\{y_i^B\}$ and $\{y_i^L\}$ such that total energy consumption is minimized while satisfying $\overline{C1}-\overline{C5}$.

Each task has different energy efficiency between clusters. A task τ_i consumes energy at the rate of $u_i^{B,max} \cdot P_{dynamic}^B$ if fully allocated on the big cluster, and $u_i^{L,max} \cdot P_{dynamic}^L$ on the LITTLE cluster. We define ef_i as τ_i 's energy efficiency ratio of the big cluster to the LITTLE cluster, expressed as

$$ef_i = \frac{u_i^{B,max} \cdot P_{\text{dynamic}}^B}{u_i^{L,max} \cdot P_{\text{dynamic}}^L}.$$
(7)

If $ef_i > 1$, executing τ_i on the LITTLE cluster is more energyefficient than the big cluster; on the contrary, if $ef_i < 1$, the converse holds. Thereby, if there is no capacity limit for each cluster, allocating all of the remaining workload of τ_i to its energy-efficient cluster consumes the least energy.

However, each cluster has its capacity limit as shown in constraints $\overline{C3}$ and $\overline{C4}$, so it might be impossible to allocate all τ_i with $ef_i > 1$ on the LITTLE cluster (or all τ_i with $ef_i < 1$ on the big cluster). Consequently, we need to rearrange each task workload allocation in order to satisfy cluster capacity limits.

We design an optimal task workload allocation algorithm based on the understanding of per-task energy efficiency on each cluster (see Algorithm 1). The task workload allocation works by two stages: 1) allocating workload in a way that consumes the minimum energy assuming infinity capacity of both clusters and 2) rearranging the workload to satisfy the feasibility conditions, especially related to cluster capacity constraints. Before stage 1), we calculate the minimum workload Algorithm 1 Optimal-Task-Workload-Allocation

1: $\tau_L \leftarrow \{\tau_i | ef_i \ge 1\}$ 2: $\tau_B \leftarrow \{\tau_i | ef_i < 1\}$ 3: Allocate $\{lo_i^B\}, \{lo_i^L\}$ according to Lemma 1 4: Allocate $y_i^L \leftarrow 1 - lo_i^B - lo_i^L, y_i^B \leftarrow 0$ for all tasks in τ^L 5: Allocate $y_i^L \leftarrow 0, y_i^B \leftarrow 1 - lo_i^B - lo_i^L$ for all tasks in τ^B 6: if Both C3 and C4 are satisfied then 7: return $\{x_i^B | x_i^B = y_i^B + lo_i^B\}, \{x_i^L | x_i^L = y_i^L + lo_i^L\}$ 8: else if Both C3 and C4 are not satisfied then 0: return task for the face blacks of the satisfied then 9: return task for the satisfied then Q٠ return not feasible 10: else if Only $\overline{C3}$ is satisfied then 11: repeat find τ_k with the closest ef_k to 1 in τ^L if $\sum_i y_i^L \cdot u_i^{L,max} - y_k^L \cdot u_k^{L,max} > m_L - \sum_i lo_i^L \cdot u_i^{L,max}$ 12: 13: $\begin{array}{l} y_k^L \leftarrow 0, y_k^B \leftarrow 1 - lo_k^B - lo_k^L \\ \tau^L \leftarrow \tau^L \setminus \{\tau_k\} \end{array}$ 14: 15: else 16: $y_k^L \leftarrow \frac{\sum_i y_i^L \cdot u_i^{L,max} - (m_L - \sum_i lo_i^L \cdot u_i^{L,max})}{u_i^{L,max}}$ 17: end if 18: 19: if C3 is violated then 20: return not feasible end if 21: **until** $\overline{C4}$ is satisfied 22: 23: else if Only C4 is satisfied then Do the corresponding process to lines 11-22. 24: 25: end if 26: return $\{x_i^B | x_i^B = y_i^B + lo_i^B\}, \{x_i^L | x_i^L = y_i^L + lo_i^L\}$

ratio (lo_i^B, lo_i^L) that should be allocated in the big and LITTLE clusters and allocate them on each cluster.

In stage 1), Algorithm 1 partitions a task set into two groups according to energy efficiency on a cluster. Let τ^B and τ^L denote a collection of tasks that are more energy-efficient when executing on the big and LITTLE clusters, respectively (lines 1–2). Then, we allocate the rest of workload (except lo_i^B, lo_i^L) of all tasks in τ^B to the big cluster and the rest of workloads of all tasks in τ^L to the LITTLE cluster (lines 4–5).

In stage 2), we check whether the allocation done by stage 1) satisfies cluster capacity constraints C3 and C4. There are 4 cases: i) if both C3 and C4 are satisfied, the allocation done by stage 1) is an energy optimal solution as well as satisfying all feasibility conditions (lines 8–9); ii) if both $\overline{C3}$ and $\overline{C4}$ are not satisfied, there is no feasible workload allocation, meaning that the task set is not feasible (lines 6–7); iii) if only C3 is satisfied, it requires to move some workload allocated on the LITTLE cluster to the big cluster until it satisfies $\overline{C4}$ (lines 10–22); and iv) if only $\overline{C4}$ is satisfied, it requires to move some workload allocated on the big cluster to the LITTLE cluster until it satisfies $\overline{C3}$ (lines 23–25). In the process of rearranging workload for cases iii) and iv), if no available big cluster capacity to accommodate more remaining workload, there is no feasible workload allocation (lines 19-21). The key issue in rearranging workload is to choose some tasks whose workload will be re-allocated. When some workload is moved from the energy-efficient cluster to the other one, energy consumption is supposed to increase. We need to move the workload in a way that the amount of increased energy consumption arising from workload migration is minimized. In addition, it should be done by the most beneficial way to become feasible. We show that choosing tasks in the order of the closest ef_i to 1 not only minimizes the amount of increased

energy consumption, but also is beneficial for feasibility in the following theorem.

We now prove that our task workload allocation algorithm achieves both feasibility-optimality and energy-optimality.

Theorem 1: Our task workload allocation presented in Algorithm 1 achieves both feasibility-optimality and energy-optimality.

Proof: We only consider the case that both clusters are turned on (i.e., $\delta_B = 1$ and $\delta_L = 1$) since in the other cases that one of the clusters is turned off, there is only one configuration of task allocation where all tasks are allocated in the cluster turned on.

Algorithm 1 works by two stages: 1) allocating workload of all tasks in τ_L to the LITTLE cluster and workload of all tasks in τ_B to the big cluster (lines 1-5) and 2) rearranging the workload to satisfy the feasibility conditions, especially related to cluster capacity constraints (lines 6-25). After stage 1), there are 4 cases: i) both $\overline{C3}$ and $\overline{C4}$ are satisfied; ii) both $\overline{C3}$ and $\overline{C4}$ are not satisfied; iii) only $\overline{C3}$ is satisfied; iv) only $\overline{C4}$ is satisfied. We first show the energy-optimality for each case of i) - iv), then, prove the feasibility-optimality later. Recall that $\tau_L = {\tau_i | ef_i \ge 1}$ and $\tau_B = {\tau_i | ef_i < 1}$. For any variable X, we denote by ΔX the amount of the variation of X through the remaining of this proof.

We now prove energy-optimality. We denote by $\Delta U^L = \sum_i \Delta u_i^L$ the amounts of workload moved from the LITTLE cluster to the big cluster, where Δu_i^L is each amount of workload of τ_i in ΔU^L . If we move ΔU^L to the big cluster, the change amount of the workload on the big cluster (denoted as ΔU^B) is calculated as

$$\begin{split} \Delta U^B &= \sum_i \Delta u_i^B = \sum_i u_i^{B,max} \cdot \Delta y_i^B \\ &= \sum_i u_i^{B,max} \cdot (-\Delta y_i^L) = \sum_i u_i^{B,max} \cdot (-\frac{\Delta u_i^L}{u_i^{L,max}}) \\ &= \sum_i -\Delta u_i^L \cdot \frac{u_i^{B,max}}{u_i^{L,max}}. \end{split}$$
(8)

Then, the amount of the variation of total energy consumption on both the big and LITTLE clusters (denoted by ΔE) is

$$\Delta E = \Delta E_L + \Delta E_B$$

$$= H \cdot P_{\text{dynamic}}^L \cdot \sum_i \Delta u_i^L + H \cdot P_{\text{dynamic}}^B \cdot \sum_i \Delta u_i^B$$

$$= H \cdot P_{\text{dynamic}}^L \cdot \sum_i \Delta u_i^L + H \cdot P_{\text{dynamic}}^B \cdot \sum_i -\Delta u_i^L \cdot \frac{u_i^{B,max}}{u_i^{L,max}}$$

$$= H \cdot \sum_i \Delta u_i^L \cdot (P_{\text{dynamic}}^L - P_{\text{dynamic}}^B \cdot \frac{u_i^{B,max}}{u_i^{L,max}})$$

$$= H \cdot P_{\text{dynamic}}^L \cdot \sum_i \Delta u_i^L \cdot (1 - ef_i)$$
(9)

Note that all variables in (9) are constant except Δu_i^L .

In case i), if $\tau_i \in \tau_L$ moves from the LITTLE cluster to the big cluster, $\Delta u_i^L < 0$ and $1 - ef_i \leq 0$, thus $\Delta E \geq 0$. If $\tau_i \in \tau_B$ moves from the big cluster to the LITTLE cluster, $\Delta E > 0$ based on the same reasoning. It means moving any

task increases E in this case. Therefore, stage 1) of Algorithm 1 is energy optimal, in case i).

In case ii), we prove that τ is not feasible later.

In case iii), we should move the specific amount of the workload from the LITTLE cluster to the big cluster, so that $\overline{C4}$ is not violated. When $\tau_i \in \tau_L$ moves from the LITTLE cluster to the big cluster, energy consumption is supposed to increase since $\Delta u_i^L < 0$ and $ef_i \ge 1$. When we move to some amount of the workload in the LITTLE cluster, choosing tasks in the order of the closest ef_i to 1 (i.e., $|1 - ef_i|$ has the minimum value) minimizes ΔE in (9). Therefore, moving tasks in the order of the closest ef_i to 1 in τ^L to the big cluster is optimal in energy consumption for case iii).

In case iv), moving tasks in the order of the closest ef_i to 1 in τ^B to the LITTLE cluster is optimal in energy consumption based on the same reasoning shown in case iii).

We now prove feasibility-optimality. We consider the following task allocation process P:

P1. allocating workload of all tasks to the LITTLE cluster P2. moving tasks in the order of the smallest ef_i from the LITTLE cluster to the big cluster until $\sum_i u_i^L = W_L$.

We prove that (a) process P minimizes $\sum_i u_i^B$ when the value of $\sum_i u_i^L$ is fixed as W_L and (b) stage 2) of Algorithm 1 satisfies the feasibility-optimality for each case of i) - iv).

Proof of (a): during process P2, when we move tasks from the LITTLE cluster to the big cluster (i.e., $\Delta U^L < 0$), according to Eq. (8),

$$\Delta U^B = \sum_{i} -\Delta u_i^L \cdot \frac{u_i^{B,max}}{u_i^{L,max}} = -\frac{P_{\text{dynamic}}^L}{P_{\text{dynamic}}^B} \cdot \sum_{i} \Delta u_i^L \cdot ef_i \quad (10)$$

For the same amount of $\Delta U^L < 0$, in order to minimize ΔU^B , we should move tasks in the order of the smallest ef_i from the LITTLE cluster to the big cluster. Therefore, (a) is true.

Proof of (b):

In case i), all feasibility conditions are already satisfied.

In case ii), by (a), moving tasks in the order of the smallest ef_i from the LITTLE cluster to the big cluster minimizes U^B but $\Delta U^B > 0$ ($\therefore \Delta u_i^L < 0 \rightarrow \Delta u_i^B > 0$). Therefore, there is no way to decrease U^L and U^B at the same time (i.e., τ is not feasible in case ii)).

In case iii), the closest ef_i to 1 in the LITTLE cluster corresponds to the smallest ef_i in the cluster since $\tau_i \in \tau_L$. By (a), moving tasks in the order of the closest ef_i to 1 in τ^L to the big cluster minimizes U^B . Therefore, if $\overline{C3}$ is violated in the process of P2 where $W_L = m_L$, τ is not feasible.

In case iv), Algorithm 1 satisfies the feasibility-optimality based on the same reasoning shown in case iii).

Therefore, Algorithm 1 satisfies feasibility/energy optimality.

Complexity. We denote by n the number of tasks in a task set. For a given combination of frequency settings, Algorithm 1 requires $O(n \log n)$ to sort a task set. Since we have only a few combinations of cluster configurations (3 for $\{\delta_B, \delta_L\}$, 9 for f_B and 5 for f_L), it takes only $O(A \cdot n \log n)$ time-complexity to solve the optimization problem in Section IV-A, where A is

a small constant. We note that according to our optimization formulation presented in Section IV-A, if a frequency combination is given, it can be solved by a Linear Programming (LP) solver. LP solvers can solve a LP formulation in polynomial time, but the polynomial is generally known as a higher degree.

V. FEASIBILITY-OPTIMAL GLOBAL SCHEDULING

While it has been proved that EDF is feasibility-optimal on a uniprocessor platform, it has been challenging to develop feasibility-optimal scheduling algorithm on a homogeneous multi-core platform. Starting from PFair [14], some feasibilityoptimal scheduling algorithms have been developed, but they are not as intuitive as EDF. Recently, a study [7] has focused on derivation of general rules that enable a scheduling algorithm to be feasibility-optimal, and therefore; the study have a significant impact on developing feasibility-optimal scheduling algorithms in that the only thing we should consider is to satisfy the general rules.

Now, the big.LITTLE multi-core processor entails the need of heterogeneous global scheduling. However, there are few studies for the scheduling; the only known study [13] introduced a feasible schedule in the process of deriving the exact feasibility condition, but it is complicated and less intuitive.

To this end, we suggest optimal scheduling rules for implicit-deadline periodic tasks running on two-type heterogeneous multi-core platforms. Greg Levin et al. [7] developed DP-Fair which guides the scheduling rules for the case of a homogeneous multi-core platform. We generalize DP-Fair to a heterogeneous multi-core platform.

DP-Fair aims at scheduling tasks by following the proportionate fairness requirement, on which each task is executed proportionally to its utilization. DP-Fair shows that imposing the fairness requirement only at job deadlines suffices to reach the optimality. It partitions time into slices based on deadlines of all jobs invoked by a task set (referred to as deadline partitioning). To ensure the fairness requirement at every deadlines, each job is assigned its execution requirement proportional to its utilization within each time slice. We note that if every job can be executed permanently at a rate equal to its utilization (referred to as a *fluid* scheduling model), the fairness requirement can be easily satisfied for all jobs. However, it is impossible to implement such a fluid schedule on practical platforms since one core cannot execute more than one task simultaneously. Thereby, DP-Fair suggests some scheduling rules for designing practical schedulers to guarantee the optimality.

We now explain how to generalize DP-Fair to a heterogeneous multi-core platform and present the scheduling rules for optimal schedulers to obey. After deadline partitioning, the k-th time slice (denoted by σ_k) is $[t_{k-1}, t_k)$ of length $l_k = t_k - t_{k-1}$.

Within the time slice σ_k , each task τ_i is then assigned its execution requirement $u_i^B \cdot l_k$, $u_i^L \cdot l_k$ on both big and LITTLE clusters, respectively. As scheduling decisions are made over time, the remaining execution of task τ_i at time t in σ_k on big and LITTLE clusters is denoted by $R_i^B(t)$ and $R_i^L(t)$, respectively. At each time t, a task is said to be a *migrating* task when its execution remaining is on both big and LITTLE clusters (i.e., $R_i^B(t) > 0$ and $R_i^L(t) > 0$), and a task is said to be an *partitioned* task when its execution remaining is solely on either big or LITTLE cluster (i.e., $R_i^L(t) = 0$ or $R_i^B(t) = 0$). A migrating task at time t can become a partitioned one whenever no execution remains either big or LITTLE cluster after t.

The major challenge to generalize DP-Fair to a heterogeneous multi-core platform is to schedule migrating tasks. If there are only partitioned tasks, we can consider each cluster as an independent homogeneous platform and apply DP-Fair in an identical way. However, if there are migrating tasks, two new issues arise: (a) we need to ensure that migrating tasks should execute its workload on at most one cluster at each time instant while they should finish all execution requirements on both big and LITTLE clusters at the end of each time slice, and (b) we need to determine which cluster executes how many migrating tasks in a way that both clusters successfully process all of the allocated workloads at every time slice.

To address issue (a), we define the task-level local laxity of τ_i at time t (denoted by $L_i(t)$) as the difference between the remaining time in a time slice σ_k and the sum of execution remaining on each cluster before the time slice, and it is presented as

$$L_i(t) = (t_k - t) - (R_i^B(t) + R_i^L(t)).$$
(11)

At the beginning of a time slice σ_k , $R_i^B(t_{k-1})$ is $u_i^B \cdot l_k$, $R_i^L(t_{k-1})$ is $u_i^L \cdot l_k$. Once a job of a task has zero task-level local laxity, it should always be executed on either the big or LITTLE cluster until the end of time slice; otherwise, the job will miss its deadline. To address issue (b), we define the cluster-level local laxity at time t, denoted by $L^B(t)$ ($L^L(t)$), as the difference between the total available capacity of the big (LITTLE) cluster from t to at the end of a time slice and the total remaining workloads on the big (LITTLE) cluster, expressed as

$$L^{B}(t) = m_{B} \cdot (t_{k} - t) - \sum_{i} R^{B}_{i}(t), \qquad (12)$$

$$L^{L}(t) = m_{L} \cdot (t_{k} - t) - \sum_{i} R_{i}^{L}(t).$$
(13)

If cluster-level local laxity of the big cluster at t (i.e., $L^B(t)$) reaches zero, all the big cores should execute jobs until the end of the time slice; otherwise, at least one job will miss its deadline due to insufficient supply. The same holds for cluster-level local laxity of the LITTLE cluster.

With the notions of task-level and cluster-level local laxities, we present our DP-Fair-Hetero scheduling rules.

Definition 1: (DP-Fair-Hetero scheduling for time slices) A scheduling algorithm belongs to DP-Fair-Hetero if it schedules jobs within a time slice σ_k according to the following rules:

- R1: Always allocate m_B jobs on the big cores at time t if its cluster-level laxity is zero (i.e., $L^B(t) = 0$), and allocate m_L jobs on the LITTLE cores at time t if its cluster-level laxity is zero (i.e., $L^L(t) = 0$);
- R2: Always run all the jobs with zero task-level local laxity (i.e., $L_i(t) = 0$);
 - R2-1: Assign partitioned tasks with zero laxity to cores prior to assigning migrating tasks with zero laxity to the rest of cores;

R3: Never run a job with no workload remaining on both clusters in the slice.

We now prove that any DP-Fair-Hetero scheduler is feasibility-optimal on two-type heterogeneous multi-core plat-forms.

Theorem 2: If a periodic implicit-deadline task set τ is feasible, any DP-Fair-Hetero scheduling algorithm always schedules the task set without any deadline miss.

Proof: The basic idea of the proof is to show that if a job of task τ_i in a task set τ misses its deadline when it is scheduled by DP-Fair-Hetero, the total workload allocated on the big or LITTLE cluster is larger than the capacity of the big or LITTLE cluster, which violates at least one of constraints C3 and C4, so that it contradicts the assumption that the task set is feasible. Details are given in Appendix B.

VI. EVALUATION

This section presents simulation results to evaluate our energy/feasibility-optimal global scheduling framework.

Simulation environment. We have three input parameters: (a) the number of cores on each big and LITTLE cluster $(m_B,$ m_L), (b) discrete frequency/voltage levels of each big and LITTLE core, and (c) individual task parameters (T_i, C_i^B, C_i^L) . We set platform-specific parameters (a) and (b), to the same as the ones employed in the ODROID-XU+E board [8], i.e., $m_B = 4, m_L = 4$, and the frequency/voltage levels as shown in Table I. Our power model with Table II is used to estimate the total energy consumption. Task sets are generated based on a technique proposed earlier [15]. For each task τ_i , T_i is uniformly chosen in [100, 1000], and C_i^L and C_i^B are chosen based on bimodal parameters in $[100, T_i]$ and $[100, C_i^L]$, respectively. For each bimodal parameter from 0.1 to 0.9 with the step of 0.1, we generate 1,500 task sets as follows. Initially, we generate a set of $m_B + 1$ tasks, and create a new set by adding a new task into the old set until the task set passes the feasibility constraints under the setting of both clusters turned on with the maximum frequencies.

We compare our global scheduling framework (annotated as Our-Global) with partitioned scheduling approaches (annotated as ILP-Partitioned). Partitioned approach statically assigns each task to a core, and task migration is not allowed. In ILP-Partitioned, task-to-core assignment is formulated by altering cluster-level task workload variables (x_i^B and x_i^L) in our optimization formulation (Energy/Feasibility-OPT) with corelevel zero-or-one variables which indicate the assignment of a task to a core. It is then solved by Integer Linear Programming (ILP).

Simulation results. To demonstrate how good global scheduling is for a big.LITTLE platform compared to partitioned scheduling in terms of both feasibility and energy consumption, we compare the number of task sets feasible by each scheduling approach and total energy consumption to schedule a feasible task set.

Figure 2 plots the number of task sets feasible by Our-Global and ILP-Partitioned with different maximum utilization on the big cluster (denoted by $U^{B,max} \stackrel{\text{def}}{=} \sum_{i} u_{i}^{B,max}$). Basically, Our-Global is generalization of ILP-Partitioned, so Our-Global dominates ILP-Partitioned. Our-Global guarantees feasi-



Fig. 2. The number of feasible task sets by $\ensuremath{\mathsf{ILP}}\xspace{\mathsf{Partitioned}}$ and $\ensuremath{\mathsf{Our-Global}}\xspace$



Fig. 3. Energy consumption ratios of ILP-Partitioned to Our-Global

ble solutions for all generated task sets while ILP-Partitioned finds 10% less feasible task sets than Our-Global.

Figure 3 plots energy consumption ratios of ILP-Partitioned relative to Our-Global with different maximum utilization on the big cluster. We note that we only show the results for feasible task sets by both approaches. We separate the generated task set into two subsets: the task sets of which all task workload is allocated only on the LITTLE cluster by Our-Global, and the others (denoted by LITTLE-only and big-LITTLE, respectively).

The LITTLE-only task sets are distributed on the range of the utilization from 0.5 to 3.5, and Our-Global makes significant differences in energy consumption compared to ILP-Partitioned for the LITTLE-only task sets. ILP-Partitioned consumes up to 60% more energy than Our-Global. This is because ILP-Partitioned decide to turn on both big and LITTLE clusters, while Our-Global only uses the LITTLE cluster in the task workload allocation. For example, we observe that there exists a case that Our-Global uses only LITTLE cluster with 500MHz but ILP-Partitioned uses both big and LITTLE clusters with 800MHz and 600MHz respectively for a task set with $U^{B,max} = 2.65$, so ILP-Partitioned consumes 87% more energy than Our-Global. Our-Global has 25% more such task sets that run on LITTLE cluster only compared to ILP-Partitioned. In general, a LITTLE core consumes less power than a big core. Thereby, if the LITTLE cluster can accommodate all workload (i.e., the big cluster is turned off), it can save much energy compared to leaving both clusters turned on.

The big-LITTLE task sets are widely spread to all the utilization distributions, and ILP-Partitioned consumes 11% more energy than Our-Global on average. Both Our-Global and ILP-Partitioned turn on both big and LITTLE clusters but different frequency settings. Our-Global can accommodate task workload with lower frequency levels. For example, there exists a case that Our-Global sets the frequency configuration of the big and LITTLE clusters to 1100MHz and 600MHz but ILP-Partitioned sets the configuration to 1500MHz and 600MHz for a task set with $U^{B,max} = 6.05$, so ILP-Partitioned consumes 35% more energy than Our-Global. A gap between Our-Global and ILP-Partitioned is increased smoothly as the utilization increases; it starts to decrease from 5.5. As the utilization increases, there is a less room for saving energy, because Our-Global and ILP-Partitioned reach to the maximum frequency for each cluster.

With the benefit of global scheduling, Our-Global can distribute all workload in the most energy-efficient way, since it not only has no restriction on task migration in contrast to ILP-Partitioned but also utilizes such a property effectively.

VII. RELATED WORK

Energy-aware real-time scheduling. In past decades, energy-aware real-time scheduling has been widely explored for both uniprocessor and multi-core platforms [16], [17]. For periodic tasks on uniprocessor platforms, Aydin et al. [11] showed that an energy-optimal schedule would execute all the tasks at a constant frequency to fully utilize the processor under the assumption that each task presents its worst-case workload behavior with the convex power consumption function.

Studies on homogeneous multi-core platforms can be classified into partitioning and global scheduling of periodic tasks. Aydin and Yang [12] addressed the problem of partitioning periodic tasks by considering both feasibility and energy consumption. They showed that the problem is NPhard and developed several heuristic algorithms by exploiting the well-known bin-packing algorithms. They experimentally showed that the Worst-Fit Decreasing (WFD) algorithm always achieves the best energy conservation when task utilization ordering is known a priori. A comprehensive survey of energyefficient partitioned scheduling under diverse task and power models with practical consideration is provided in [16], [17]. On the contrary to partitioned scheduling, there are some energy-efficient global scheduling algorithms with guarantees of feasibility [18], [19], [20], [21]. Funaoka et al. [18] proposed real-time static voltage and frequency scaling (RT-SVFS) techniques based on an optimal real-time scheduling algorithm for homogeneous multi-core platforms. The techniques are regarded as a static voltage/frequency scaling approach, because after setting the initial voltage and frequency it will not change during run-time. The techniques have been proven optimal when the voltage and frequency can be controlled both uniformly and independently among processors. Based on RT-SVFS, real-time dynamic voltage and frequency scaling (RT-DVFS) was presented in order to accommodate to dynamic environments [19].

While a lot of research has been done for homogeneous multi-core platforms, comparatively less research has been done for heterogeneous multi-core platforms. Moreover, partitioning approach is only considered in the literatures [3], [4], [5], [6]. Yu and Prasanna [3] addressed the problem of assigning periodic tasks on heterogeneous platforms with the setting of the frequency level per task. They formulated the problem as an Integer Linear Programming (ILP) to minimize the energy consumption and provided a linear relaxation heuristic algorithm. The other related works [4], [5], [6] studies energyefficient task partitioning on platforms with a static frequency for each processor. Chen and Thiele [4] provided a fully polynomial-time approximation scheme based on dynamic programming for a case of two-type heterogeneous processors. This work was later extended in [5] for *n*-type heterogeneous processors. Chen et al. [6] formulated the energy-efficient task partitioning problem as an ILP and provided polynomial-time algorithms by applying existing bin-packing algorithms based on a relaxation of some assumptions. The most of related work for heterogeneous multi-core platforms considers task partitioning approach where task migration is not allowed, and it is proved to be NP-hard in a strong sense. Thus, they focused on developing efficient heuristic algorithms with approximation bounds. This paper focuses on energy/feasibility optimal global scheduling on two-type heterogeneous platforms. We develop an optimal task workload allocation algorithm from both the feasibility and energy consumption points of view and establish optimal scheduling rules for two-type heterogeneous multi-core platforms.

Real-time scheduling on heterogeneous multi-core platforms. From the feasibility point of view, the scheduling problem on heterogeneous multiprocessors has been studied in the past [22], [23], [24], [13]. Baruah [22] considered the task partitioning problem which determines whether the tasks can be partitioned among processors in such a manner that all timing constraints are met. Andersson et al. [23] proposed a task assignment heuristic algorithm for the two-type platforms. Raravi et al. [24] considered an intra-migrative scheduling problem, which statically assigns each task to a core type and allows task migration among cores of the same core type, and proposed a linearithmic task assignment algorithm. For global scheduling, Baruah [13] provided an exact feasibility analysis.

VIII. CONCLUSION

This paper is motivated by an attempt to see how good global scheduling, beyond partitioned scheduling, can be for big.LITTLE platforms (one of the cutting-edge heterogeneous multi-core architectures) in the perspective of both core utilization and energy saving. To this end, we develop an energy/feasibility-optimal global scheduling framework which determines big.LITTLE platform configurations and global job schedules, so that the energy consumption is minimized without compromising feasibility. Moreover, we suggest DP-Fair-Hetero as optimal scheduling rules for implicit-deadline periodic tasks running on two-type heterogeneous multi-core platforms. This work will be a basis for designing efficient global schedulers for heterogeneous multi-core platforms.

One of the major concerns on global scheduling is migration overhead. Hence, a direction of our future work includes incorporating migration overhead into our framework and developing an efficient global scheduling algorithm targeting for general heterogeneous multi-core platforms.

REFERENCES

- R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," ACM Computing Surveys, vol. 43, pp. 35–44, 2011.
- [2] ARM, "big.little technology: The future of mobile," 2013. [Online]. Available: http://www.arm.com/files/pdf/ big-LITTLE-Technology-the-Futue-of-Mobile.pdf
- [3] Y. Yu and V. K. Prasanna, "Resource allocation for independent realtime tasks in heterogeneous systems for energy minimization," *Journal* of Information Science and Engineering, vol. 19(3.
- [4] J.-J. Chen and L. Thiele, "Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements," in *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2008)*, 2008.
- [5] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in *Proceedings of the Conference and Exhibition of Design, Automation, and Test in Europe (DATE 2009)*, 2009.
- [6] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2009)*, 2009.
- [7] G. Levin, F. Shelby, S. Caitlin, P. Ian, and B. Scott, "DP-Fair: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS*, 2010.
- [8] Hardkernel co. Ltd., "ODROID-XU+E specification," 2014. [Online]. Available: http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu
- [9] A. Carroll and G. Heiser, "Unifying DVFS and offlining in mobile multicores," in *RTAS*, 2014.
- [10] Hardkernel co. Ltd., "ODROID-XU+E," 2013. [Online]. Available: http://www.hardkernel.com/main/products/prdt-info.php?g-code= G137463363079
- [11] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of IEEE Real-Time Systems Symposium*, 2001.
- [12] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.
- [13] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multirpocessor platforms," in *RTSS*, 2004.
- [14] S. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: a notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, 1996.
- [15] T. Baker, "An analysis of EDF schedulability on a multiprocessor," *IEEE Transactions on Parallel Distributed Systems*, vol. 16, no. 8, pp. 760–768, 2005.
- [16] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *RTCSA*, 2007.
- [17] D. Li and J. Wu, Energy-aware Scheduling on Multiprocessor Platforms. SpringerBriefs in Computer Science, 2013.
- [18] K. Funaoka, S. Kato, and N. Yamasaki, "Work-conserving optimal realtime scheduling on multiprocessors," in *ECRTS*, 2008, pp. 13–22.
- [19] K. Funaoka, A. Takeda, S. Kato, and N. Yamasaki, "Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors," in *Proceedings of the International Symposium on Industrial Embedded Systems (SIES 2008)*, 2008.
- [20] D.-S. Zhang, F.-Y. Chen, H.-H. Li, S.-Y. Jin, and D.-K. Guo, "An energy-efficient scheduling algorithm for sporadic real-time tasks in multiprocessor systems," in *Proceedings of the IEEE International Conference on High Performance Computing and Communications* (HPCC 2011), 2011.
- [21] G. A. Moreno and D. de Niz, "An optimal real-time voltage and frequency scaling for uniform multiprocessors," in *RTCSA*, 2012.
- [22] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *RTAS*, 2004.
- [23] B. Andersson, G. Raravi, and K. Bletsas, "Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors," in *RTSS*, 2010.
- [24] G. Raravi, B. Andersson, K. Bletsas, and V. Nelis, "Task assignment algorithms for two-type heterogeneous multiprocessors," in *ECRTS*, 2012.

We calculate the minimum workload ratios induced by constraint C2.

we calculate the minimum workload ratios induced by constraint C2. In the case of $u_i^{L,max} > 1$ and $u_i^{B,max} > 1$, τ is not feasible. In the case of $u_i^{L,max} > 1$ and $u_i^{B,max} \leq 1$, $x_i^{L} \cdot u_i^{L,max} + x_i^{B} \cdot u_i^{B,max} \leq 1$ $\iff (1 - x_i^B) \cdot u_i^{L,max} + x_i^B \cdot u_i^{B,max} \leq 1$ (\because C1) $\iff \frac{u_i^{L,max} - 1}{u_i^{L,max} - u_i^{B,max}} \leq x_i^B$ $(u_i^{L,max} > 1, u_i^{B,max} \leq 1 \rightarrow 0 < \frac{u_i^{L,max} - 1}{u_i^{L,max} - u_i^{B,max}})$ $\therefore lo_i^B = \frac{u_i^{L,max} - 1}{u_i^{L,max} - u_i^{B,max}}$ Additionally, $u_i^{B,max} \leq 1 \rightarrow 0 \leq x_i^L$ $(\because \tau_i$ can integrally be executed on the LITTLE cluster). In the case of $u_i^{L,max} - u_i^{L,max} \leq x_i^L, 0 \leq x_i^B$ Otherwise, i.e. $u_i^{L,max} \leq 1$ and $u_i^{B,max} \leq 1$, τ_i can integrally be executed on either the big or LITTLE

 τ_i can integrally be executed on either the big or LITTLE clusters. Therefore, $0 \le x_i^L, x_i^B$

APPENDIX B - PROOF OF THEOREM 2

We prove this theorem by contradiction. Suppose a job of task τ_i misses its deadline when it is scheduled by DP-Fair-Hetero. Then, there must exist the time slice σ_k and the earliest time t' in σ_k such that the job of τ_i has task-level local laxity of -1 at t' (i.e., $L_i(t') < 0$, $t_{k-1} \le t' < t_k$). The job became zero laxity at t' - 1, so it should be executed at t' - 1 according to R2. However, it fails to execute at t' - 1. We denote by $\tau' = \{\tau_j | L_j(t'-1) = 0\}$ the task set of which a job has zero laxity at t' - 1.

We consider two cases: (A) $|\tau'| > m_B + m_L$, (B) $|\tau'| \le$ $m_B + m_L$.

Case A: If $|\tau'| > m_B + m_L$, then the sum of remaining execution on both the big and LITTLE clusters in $[t'-1, t_k)$ is larger than $(m_B + m_L) \cdot (t_k - (t' - 1))$ since the remaining execution of a zero laxity job at t'-1 is $t_k - (t'-1)$. Moreover, all cores in the big and LITTLE clusters should be busy in interval $[t_{k-1}, t'-1)$. If there is an idle core at t'' in $[t_{k-1}, t'-1]$ 1), the only reason that a job of $\tau_i \in \tau'$ cannot be executed on the idle core is that the job is already executed on the other core at t''. It means $|\tau'| \leq m_B + m_L$, and this contradicts the assumption that $|\tau'| > m_B + m_L$. Therefore, the sum of the total workload allocated on each cluster in $[t_{k-1}, t'-1)$ and $[t'-1,t_k)$ (i.e. $[t_{k-1},t_k)$) is larger than the sum of the total capacity of each cluster in $[t_{k-1}, t_k)$, meaning that at least either $\overline{C3}$ or $\overline{C4}$ are violated. This contradicts the assumption that τ is feasible.

Case B: We consider the case that $|\tau'| \leq m_B + m_L$. The deadline miss job of τ_i was not executed at t'-1 even though the job has zero laxity, which implies that there is no job to execute except zero laxity jobs. Moreover, by assumption, the number of zero laxity jobs executed at t' - 1 should be less than $m_B + m_L$. Therefore, if $|\tau'| \leq m_B + m_L$, there must exist at least one idle core at t' - 1.

(Case B-1): In the case that task τ_i is a migrating task at t'-1 (i.e., $R_i^B(t'-1) > 0$ and $R_i^L(t'-1) > 0$), a job of τ_i should be executed on the idle core at t' - 1. This contradicts that the job of τ_i has task-level local laxity of -1 at t'.

(Case B-2): In the case that task τ_i is a partitioned task at t'-1 (i.e., either $R_i^L(t'-1) = 0$ or $R_i^B(t'-1) = 0$). Without loss of generality, we assume $R_i^L(t'-1) = 0$.

If the big cluster has an idle core at t' - 1, a job of task τ_i should be executed at t' - 1 by R2. It contradicts that the job of τ_i has task-level local laxity of -1 at t'.

If the big cluster has no idle core at t' - 1, there must be more than m_B partitioned zero laxity jobs on the big cluster at t'-1 (If not, a job of τ_i should be executed at t'-1 by R2-1). We define $\tau'' = \{\tau_j | L_j(t'-1) = 0 \land R_j^L(t'-1) = 0\}$. The sum of the remaining execution of all jobs of $\tau_j \in \tau''$ in $[t'-1, t_k)$ should be larger than $m_B \cdot (t_k - (t'-1))$. Since $L^B(t'-1) < 0$, there must exist the latest time $t'' \in [t_{k-1}, t'-1)$ such that $L^B(t'') = 0$ (if not, $\forall t \in [t_{k-1}, t_k), L^B(t) < 0 \to \tau$ violates $\overline{C3}$). Then, there must be at least one idle core at t'' because $\forall t > t'', L^B(t) < 0$. The only reason that a job of $\tau_j \in \tau''$ cannot be executed on the idle core is that the job is already executed on the other core at t''. If such job of τ_j is a migrating one at t'', it must be executed on the big cluster by R1. If such job of τ_j is a partitioned one at t'', it is executed on the big cluster. Thus, all jobs of tasks in τ'' are executed on the big cluster, which implies $|\tau''| \leq m_B$, and this contradicts $|\tau''| > m_B.$

This also holds for the case that $R_i^B(t'-1) = 0$.

Therefore, for all cases, we show that if DP-Fair-Hetero fails to schedule a task set, the task set should be not feasible.