# Energy-Aware Real-Time Scheduling Algorithm on ARM big.LITTLE HMP Architecture

Insu Jang and Jaehun Roh
Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea
bluewave8375@naver.com, shwogjs3@naver.com

*Abstract*—**As many systems including running in real-time environment are embedded devices using ARM architecture based application processors, architecture considering algorithms for managing resources are very important. The cutting edge technology of the ARM architecture is big.LITTLE heterogeneous multicore processing (HMP). However, there is few studies about real-time scheduling based on this architecture. To solve this problem, we propose a new workload assignment algorithm called VFS-Hetero-Split by extending the existing Hetero-Split with more consideration on properties of the big.LITTLE HMP architecture, such as up-threshold, dynamic voltage frequency scaling (DVFS), and so on. It can assign workloads onto big and LITTLE clusters with the same time complexity of the existing Hetero-Split algorithm but total required power consumption is less than that of Hetero-Split. With VFS-Hetero-Split, running generated workload assignment with each target frequency can successfully reduce power consumption up to 48% without losing any feasibility compared to that running with the maximum frequency.**

***Keywords—big.LITTLE; heterogeneous multicore processing; real-time scheduling; dynamic voltage frequency scaling***

## I. Introduction

In these days, many embedded systems using ARM architecture based application processors for power efficiency are released. The earlier version of ARM architecture is much power-efficient, however, this power efficiency is being decreased because of growth of performance demand. Therefore, ARM introduced a big.LITTLE architecture to satisfy performance demand while keeping power efficiency. At first, all big.LITTLE runs as cluster migration. As we cannot use both cluster, it is less efficient and less flexible. To solve this problem, ARM introduced a new algorithm of big.LITTLE, called heterogeneous multicore processing (HMP). Under HMP scheduling algorithm, all cores can run simultaneously, so it became very flexible. To determine whether this task should be run on the big cluster or on the LITTLE cluster, up-threshold and down-threshold is newly denoted considering power efficiency based on the current demand. During dynamic voltage frequency scaling (DVFS), the target frequency of the task is increasing and finally the task is migrated onto the big cluster when the frequency is larger than up-threshold.

Like this, we have to consider different characteristics between big and LITTLE cluster, so scheduling on a heterogeneous multicore architecture is challenging than scheduling on identical multicore architecture. Especially, HMP architecture is the cutting-edge technology of heterogeneous multicore architecture, there are few studies about scheduling algorithm on HMP architecture even demand is growing. Therefore, we introduce a new real-time scheduling algorithm running on ARM big.LITTLE architecture. We extend one of fully-migrative heterogeneous multicore real-time scheduling algorithms, Hetero-Split, to VFS-Hetero-Split considering characteristics of HMP architecture such as DVFS, workload decomposition, threshold, and so on [1]. Satisfying properties of Hetero-Split, a workload assignment generated by VFS-Hetero-Split can be used with Hetero-Wrap algorithm, the first optimal two-type heterogeneous multicore scheduling algorithm.

The remaining paper consists of the following sections: section II introduces background knowledge about ARM big.LITTLE model, power model of this architecture, DVFS, workload decomposition, and our system model. In section III, we introduce our algorithm VFS-Hetero-Split. We adopted similar approach that Hetero-Split used: list feasibility conditions and consider one by one. To make a task set satisfy these conditions, we consider workload decomposition, DVFS, task migration, and so on. In section IV, we perform experiment how power consumption is reduced compared to that when processor is running with its maximum frequency using the simulator that we implemented.

## II. Backgrounds

### A. ARM big.LITTLE model

Heterogeneity means that it is composed of different types of cores. One of the famous heterogeneous architectures is big.LITTLE architecture suggested by ARM holdings, one of the most famous application processor design company.

As several years ago, architectures designed by ARM did not require high power. However, as performance of chips is increasing, required power consumption is also increasing, which is too burden to handle with the current battery technology. To solve this problem, ARM implemented a new concept of heterogeneous multicore architecture, named big.LITTLE. Although big and LITTLE cores share the same Instruction-Set-Architecture (ISA), big cores and LITTLE cores have many differences such as a predictor and issue bandwidth between them. It means that they have different power efficiency and performance characteristics. Big cores have a complex architecture but have good performance and high power consumption, while LITTLE cores are good for power efficiency due to their simple architecture. Therefore, if a task is high-demand, it is migrated to a big core to meet their demand or runs in a LITTLE core if not. Table 1 shows differences of

A53 architecture used as LITTLE cores and A57 architecture used as big cores [4].

TABLE I. SPECIFICATIONS OF THE A57 AND A53 ARCHITECTURES

| | **A53 architecture** | **A57 architecture** |
|---|---|---|
| Decode | 2-wide | 3-wide |
| Pipeline depth | 8 | 15 |
| Instruction order | In-order | Out-of-order |
| branch prediction | Conditional & indirect branch prediction | two-level |
| Execution ports | 2 | 8 |
| L1 cache (KiB) | 8 to 64(I) + 8 to 64(D) | 48(I) + 32(D) |
| L2 cache (KiB) | 128 to 2048 | 512 to 2048 |
| DMIPS/MHz | 2.3 | 4.1 to 4.76 |
| big.LITTLE role | LITTLE | Big |

When the big.LITTLE concept is released at first, big.LITTLE architectures are executed as a cluster migration which uses only either a big cluster or a LITTLE cluster at the same time. In normal situation, only the LITTLE cluster is used while all tasks in the LITTLE cluster are migrated and run in the big cluster when they are being heavy. It is simple to implement but less efficient and flexible because we can only use one cluster at a time. To solve this problem and to increase efficiency, Heterogeneous Multicore Processing (HMP), also called as Global Task Scheduling (GTS), is introduced [2, 3]. Since all physical cores can be controlled simultaneously, power management can be done more carefully and efficiently.

Nowadays many chip vendors such as Samsung and Qualcomm are making ARM big.LITTLE processors. In case of Samsung Exynos 7420, it is composed of four A53 LITTLE cores and four A57 big cores and acts as HMP. Figure 1 and Figure 2 are the number of cores – frequency – power consumption curve which are measured by Andrei [5].

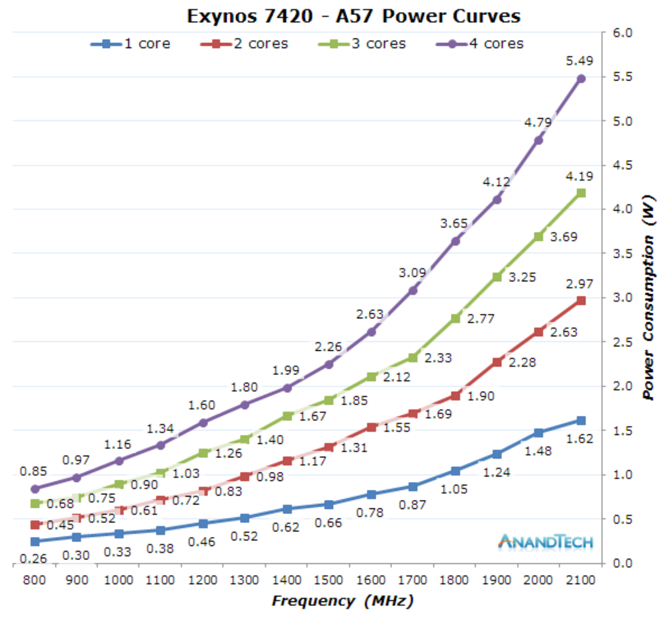

Fig. 1. The power curve of A57 big cores in Samsung Exynos 7420

Figure 1 is the graph about the power consumption of A57 big cores in Samsung Exynos 7420 processor. The power curve has different shape from A15 architecture based big cores in Samsung Exynos 5410 [6], because Samsung Exynos 7420 runs as heterogeneous multicore processing, not cluster migration processing. As cores can be controlled individually, the total power consumption is proportional to the number of cores.

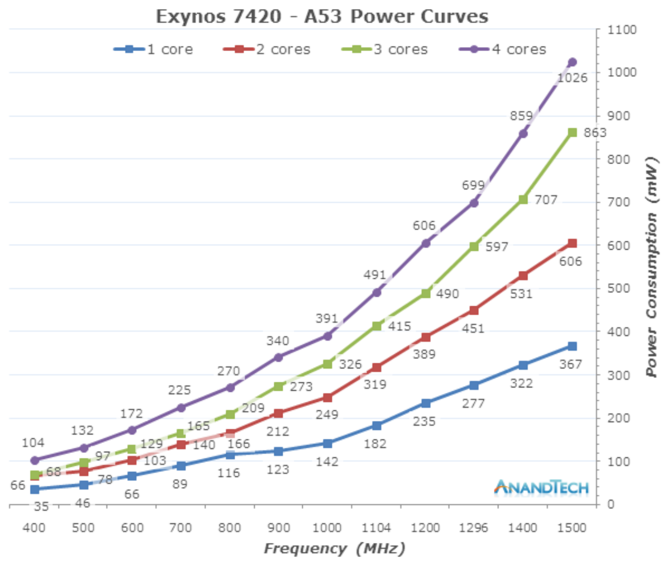

Fig. 2. The power curve of A53 LITTLE cores in Samsung Exynos 7420

Figure 2 is the graph about power consumption of LITTLE cores. It has a similar shape with what of big cores, while the LITTLE cores spend less power than big cores at the same frequency.

## *B. Power and energy model*

To assign tasks in terms of power efficiency, we have to model CPU power consumptions. We applied the model that Aaron used and distinguished the state into active, idle, and offline [7]. Active state is a state when CPU is on and works some task. Idle state is a state when CPU is on, but they do not have some task to do. Offline state means that the core is in sleep state so cannot do any works before going into idle state. Equation 1 is the represents total power consumption of CPU.

$$P = P_{uncore} + n(P_{static} + P_{dynamic}) \tag{1}$$

where $n$ is the number of cores that are not in offline state.

$P_{uncore}$ means the power consumption which is independent with the state like idle, active. Even every cores are in off state, the amount of power $P_{uncore}$ should be consumed. $P_{static}$ means the power consumption of a CPU core when it is on but idle. This is independent from the workload but affected by the voltage of the core. $P_{dynamic}$ is the power consumption when the core is online and handles some tasks. The following equations represent $P_{static}$ and $P_{dynamic}$.

$$P_{static} = C_{static} * V \tag{2}$$

$$P_{dynamic} = C_{eff} * f * V^2 \tag{3}$$

where $C_{static}$ is a constant depending on the core type, $C_{eff}$ is an effective capacitance, $f$ is the core frequency, and $V$ is the core voltage.

Let time to complete an i-th task be $T_i$. Then total energy consumption $E_i$ to complete an i-th task should be:

$$E_i = P_i * T_i \tag{4}$$

## C. DVFS and workload decomposition

Dynamic Voltage Frequency Scaling (DVFS) is a technique used in CPUs to reduce power consumption. Especially, as Aaron said, most of devices are embedded devices and have limited power budget so energy efficiency is one of primary design criterions.

As explained in the power model section, an additional required power consumption to handle a task when the core is in idle state can be expressed as [8]:

$$P_{add} = C_{eff} f V^2 \tag{5}$$

According to the DVFS table [9], the core voltage is asymptotically proportional to the core frequency. Therefore, we can re-express the above equation as:

$$P_{add} \cong C_{eff} f^3 \tag{6}$$

We simply assumed that the frequency is the number of instructions that the core can handle in a time unit and the workload of the task is the number of instructions to be completed in our system model. Thus, the execution time can be expressed as:

$$T_i = \frac{W_i}{f} \tag{7}$$

where $w_i$ is workload of the task $\tau_i$. Then, total power consumption to process a task can be expressed as:

$$E_i = P_{add} * T_i \propto f^2 \tag{8}$$

which means that the required amount of energy proportional to square of the frequency. Therefore, rather than finishing tasks fast with high frequency, it is better to run tasks with a frequency as low as possible. In real-time environment, every tasks have its own deadline. Hence, we define the target frequency that minimizes power consumption:

$$f_{target,i} = \frac{W_i}{T_i} = \frac{W_i}{P_i} = \frac{W_i}{D} \tag{9}$$

Note that we increased the execution time of i-th task as long as possible, to its deadline. As all tasks have its own unique target frequency, cores should adjust their running frequency and appropriate voltage whenever switching cores, which is why it is called DVFS.

However, even we set frequency double, the execution time will not be half, according to Amdahl's law [10]. We must consider this characteristic so decompose workload of i-th task into two parts: on-cpu workload $W_i^{on}$ and off-cpu workload $W_i^{off}$ [11]. Then we can view the execution time of i-th task as:

$$T_i = T_i^{on} + T_i^{off} = \frac{W_i^{on}}{f} + \frac{W_i^{off}}{r} \tag{10}$$

where $r$ is the speed of another devices except for CPU and $T_i^{off}$ is the execution time to handle off-cpu workload $W_i^{off}$ and constant regardless of the cpu frequency $f$. Then, the target frequency of i-th task can be recalculated as:

$$f_{target,i} = \frac{W_i^{on}}{P_i - T_i^{off}} = \frac{W_i^{on}}{D - T_i^{off}} \tag{11}$$

Note that the new target frequency is less than or equal to the old target frequency, i.e. $\frac{W_i^{on}}{D-T_i^{off}} \leq \frac{W_i}{D}$, which means that we can set lower frequency and reduce more power consumption when considering workload decomposition.

*Proof:* With workload decomposition, $D = T_i = T_i^{on} + T_i^{off}$ and $W_i = W_i^{on} + W_i^{off}$. Then the inequality can be expressed as $W_i^{on}\left(T_i^{on} + T_i^{off}\right) \leq \left(W_i^{on} + W_i^{off}\right)T_i^{on}$, so $T_i^{on}T_i^{off}(f - r) \geq 0$. According to memory hierarchy [12], the fastest component in computers is a processor. Therefore, the inequality always holds. ■

Following the definition of big.LITTLE execution ratio $r_i$, big cores are $r_i$ times faster than LITTLE cores for i-th task, i.e. $T_i^{big,on} = \frac{1}{r_i} T_i^{LITTLE,on}$. Hence, the target frequency for big core and LITTLE core of i-th task have the following relationship:

$$f_{target,i}^{big} = \frac{1}{r_i} f_{target,i}^{LITTLE} \tag{12}$$

## D. System model

We consider an ARM big.LITTLE HMP architecture based cluster $\pi$ is organized with a Cortex-A57 big cluster $\pi^{big}$ with $n^{big}$ number of big cores and a Cortex-A53 LITTLE cluster $\pi^{LITTLE}$ with $n^{LITTLE}$ number of LITTLE cores. The range of frequency depends on the type of the cluster : a big core can set its frequency $f^{big}$ between $(f_{min}^{big}, f_{max}^{big})$ and so does a LITTLE core between $(f_{min}^{LITTLE}, f_{max}^{LITTLE})$. Although some instructions require more than one number of cycles to complete, we simply assume that the core frequency is the number of instructions that the core can handle in a time unit.

Each task $\tau_i \in \tau$ is a periodic task and has 4 characteristics:

- A period $P_i$,
- The amount of workload $W_i$ denoted by the number of instructions in the task,
- On-cpu workload ratio $r_i^W$ that represents the portion of on-cpu workload when assuming total workload as one,
- big.LITTLE execution ratio $r_i^T$ that represents the execution ratio in a big core to a LITTLE core when running in the same frequency.

To match our assumption about the cluster, a period $P_i$ is defined as the number of time unit to complete this task and the amount of workload $W_i$ is denoted by the number of instructions in this task. Note that big.LITTLE execution ratio is always larger than one because a big core is always faster than a LITTLE core when they run with the same frequency.

Our assumptions about the tasks are all same with what Chwa assumed. Therefore, it our algorithm can successfully generate a workload assignment that has the same properties with what is generated by Hetero-Split algorithm, we can use Hetero-Fair guidelines and Hetero-Wrap algorithm as they are.

## III. VFS-Hetero-Split : Task Workload Assignment

In this section, we introduce our approach to assign task workloads on ARM big.LITTLE architecture. Our approach is extended from Hetero-Split and considers not only feasibility but also energy-efficiency.

Hetero-Split algorithm approached to the problem with two steps: feasibility conditions and per-cluster task workload assignment algorithm. We approach to the problem with the same method, but including an additional feasibility condition and a revised task workload assignment algorithm for ARM big.LITTLE considering energy efficiency.

### *A. Feasibility conditions*

According to feasibility analysis of fully-migrative heterogeneous multicore scheduling [13], five conditions C1-C5 must hold for a task workload assignment to be feasible. Among these constraints, Hetero-Split aimed to find a “specific” feasible task workload assignment by only focusing on C2 (deadline constraint) and C3 and C4 (capacity constraint) as follows:

$$\text{C2} : \forall \tau_i \in \tau, u_i^1 + u_i^2 \le 1,$$

$$\text{C3} : \textstyle\sum_{\tau_i \in \tau} u_i^1 \le m_1,$$

$$\text{C4} : \textstyle\sum_{\tau_i \in \tau} u_i^2 \le m_2.$$

where $u_i^k$ is the utilization of the i-th task when running on a type-k cluster and $m_k$ is the total capacity of the type-k cluster.

In our model, these conditions can be regarded as:

$$\text{C2'} : \forall \tau_i \in \tau, T^{big}_{f=f^{big}_{target,i},i} + T^{LITTLE}_{f=f^{LITTLE}_{target,i},i} + T_i^{off} \le P_i,$$

$$\text{C3'} : \textstyle\sum_{\tau_i \in \tau} u^{big}_{f=f^{big}_{target,i},i} \le m_{big},$$

$$\text{C4'} : \textstyle\sum_{\tau_i \in \tau} u^{LITTLE}_{f=f^{LITTLE}_{target,i},i} \le m_{LITTLE}.$$

where the utilization can be simply calculated as:

$$u_{f,i}^{LITTLE} = \frac{W_i^{on,LITTLE}}{f} * \frac{1}{P_i - T_i^{off}}, u_{f,i}^{big} = \frac{u_{f,i}^{LITTLE}}{r_i} \qquad (13)$$

Note that C2 is still the deadline constraint, but expressed with the execution time, not with utilization because of workload decomposition.

We are now considering DVFS, so there is an another constraint named frequency constraint. If the target frequency of a task that satisfies the deadline of the task is larger than the maximum frequency $f_{max}$, it means it cannot run on that core. Therefore, the new frequency constraint C1 is:

$$\text{C1} : \forall \tau_i \in \tau, f^{big}_{target,i} \le f^{big}_{max}$$

Note that there is no restriction about the target frequency of a LITTLE core because a big core is always faster than a LITTLE core and has larger target frequency, i.e. $f^{big}_{max} \ge f^{LITTLE}_{max}$. The i-th task might not be feasible solely on LITTLE cluster, but is always feasible on the whole cluster when it satisfies C1.

### *B. ARM big.LITTLE workload assignment algorithm*

We implement improved Hetero-Split, called VFS-Hetero-Split. It guarantees to find a feasible workload assignment on ARM big.LITTLE architecture satisfying new constraints C1-C4, similar to Hetero-Split. At first, we consider the deadline constraint, and then assign tasks into clusters with several new definitions from HMP architecture.

#### *1) Considering the deadline constraint*

Different from the existing Hetero-Split algorithm, VFS-Hetero-Split algorithm does not consider workload fraction at this time. Instead, we calculate the target frequency for a big core and a LITTLE core of each task, i.e. $f^{big}_{target,i}$ and $f^{LITTLE}_{target,i}$, by using equation 11 and equation 12.

If the target frequency for a big core is larger than the maximum frequency of a big core, i.e. $f^{big}_{target,i} > f^{big}_{max}$, then this task should violate either the deadline constraint or the frequency constraint, which means it is not feasible.

Also, if the target frequencies for either big core or LITTLE core is less than the minimum frequencies of the corresponding cluster, then set the target frequency with the minimum frequency.

We calculated the target frequencies of all tasks in the task set which satisfy their deadline constraints. Only what we have to do is to assign these tasks onto big.LITTLE clusters considering power consumption and capacity constraints.

#### *2) Considering up-threshold*

In HMP, there are definitions about up-threshold and down-threshold. Up-threshold is a frequency for a LITTLE core which is a match point between energy efficiency of the LITTLE core and that of a big core. In opposite, down-threshold is a frequency for a big core which is a match point between energy efficiency of a LITTLE core and the big core [14].

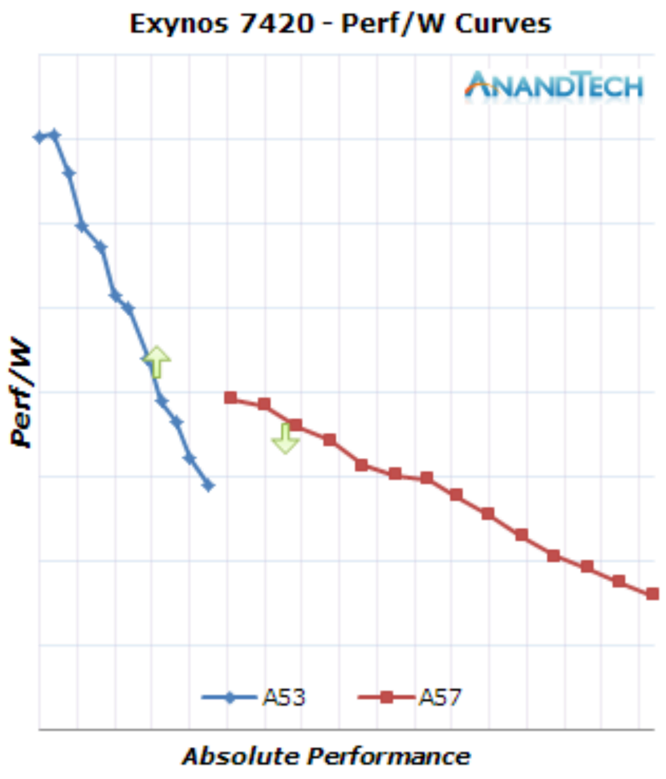

Fig. 3. Energy efficiency – performance graph of Samsung Exynos 7420

Figure 3 is a performance-energy efficiency graph of Samsung Exynos 7420 measured by Andrei. The up arrow on the blue line is up-threshold for an A53 LITTLE core and the down arrow on the red line is down-threshold for an A57 big core. In existing HMP scheduling, we don't know the actual amount of tasks so frequency is changed dynamically considering a current demand in each CPU clock cycle. Once the frequency for a LITTLE core calculated from the current demand become larger than up-threshold, it is migrated to a big core.

Slightly different from the original use of up-threshold, we use only up-threshold to figure out which cluster is more energy-efficient for tasks to be run on. As VFS-Hetero-Split allocates tasks by using Hetero-Wrap algorithm, which is one of offline task allocation method, we don't have to use up-threshold and down-threshold in terms of the current demand of tasks because we know which cluster is more energy efficient for each task before actually assigning it. If the target frequency for a LITTLE core of a task is larger than up-threshold, it means it is more energy-efficient when it runs on a big core.

To assign tasks in terms of energy efficiency, we should first calculate up-threshold for each task. As the exact calculation of up-threshold is too complex, some factors will be ignored during calculation. By definition of up-threshold, the following equation holds:

$$E^{LITTLE}_{f=f_{up,i}} = E^{big}_{f=f^{big}_{target,i}} \tag{14}$$

where $f_{up,i}$ is up-threshold.

By the equation 4 with workload decomposition,

$$P^{LITTLE}_{core,i} * T^{LITTLE}_{f=f_{up,i}} + E^{off}_{i} = P^{big}_{core,i} * T^{big}_{f=f^{big}_{target,i}} + E^{off}_{i} \tag{15}$$

According to our power model, $P$ can be expressed as:

$$\left(P^{LITTLE}_{static} + C^{LITTLE}_{eff} * f_{up,i} * V^{2}_{up,i}\right) * \frac{W^{on}_{i}}{f_{up,i}}$$
$$= \left(P^{big}_{static} + C^{big}_{eff} * f^{big}_{target,i} * V^{big}_{target,i}{}^{2}\right) * \frac{W^{on}_{i}}{f^{big}_{target,i}} \tag{16}$$

As $P_{static}$ is proportional to the core voltage and the voltage is asymptotically proportional to the core frequency, then

$$\left(C^{big}_{static} * f^{big}_{target,i} + C^{big}_{eff} * f^{big}_{target,i}{}^{3}\right) * \frac{W^{on}_{i}}{f^{big}_{target,i}}$$
$$\approx \left(C^{LITTLE}_{static} * f_{up,i} + C^{LITTLE}_{eff} * f^{3}_{up,i}\right) * \frac{W^{on}_{i}}{f_{up,i}} \tag{17}$$

By calculating equation, up-threshold of i-th task can be estimated as:

$$f_{up,i} = \sqrt{\frac{C^{big}_{static} - C^{LITTLE}_{static} + C^{big}_{eff} * f^{big}_{target,i}{}^{2}}{C^{LITTLE}_{eff}}} \tag{18}$$

If $f^{LITTLE}_{target,i} > f_{up,i}$, then assign an i-th task into big cluster. If not, assign it into LITTLE cluster. In this step, if the the i-th task has $f^{LITTLE}_{target,i} > f^{LITTLE}_{max}$, assign it into big cluster without any doubt. Up-threshold is calculated by the target frequency of a big core that satisfies deadline constraint and frequency constraint, it violates neither of them.

### 3) *Considering DVFS*

After assigning them, it still might not satisfy capacity constraint. Then, we can make a task set feasible by increasing running frequencies of tasks in it. In this situation, deadline constraint will never violate because the core frequency is anti-proportional to execution time, so execution time will decrease when the frequency increases.

As we can see in our power model, required amount of power for doing the same amount of workload is proportional to square of the core frequency. Therefore, we have to consider not only the mean of running frequencies of tasks, but also the variance of those when increasing tasks' running frequency.

*Proof:* let there are two tasks and both have the same deadline and execution time half deadline at the core frequency $f$. So the sum of their execution time is equal to the capacity of a core, i.e., utilization of the core is 1. Suppose that there is no off-cpu workload and no static or uncore power consumption for the core. Then a total power consumption $E$ can be expressed as:

$$E = 2C^{core}_{eff} * f * V^{2} * T = 2C^{core}_{eff} * f^{2} \tag{19}$$

where $T$ is the execution time, just simply assumed $T = \frac{1}{f}$ and the core voltage is also simplified as $V = f$.

This task set is feasible. if we decrease running frequency of one task by half and increase running frequency of the other by 1.5 times, it still be feasible and the total utilization of the core is unchanged. However, as required power is proportional to square of the frequency, the total power consumption changes to $E'$ as follows:

$$E' = C_{eff}^{core} * \left\{(0.5f)^3 * 2t + (1.5f)^3 * \frac{2}{3}t\right\} = \frac{5}{4}E \quad (20)$$

which means total power consumption is increased when the variance of running frequencies is increased from 0 to 0.25. As variance is always larger than or equal to 0, the total power consumption is larger than or equal to that with variance = 0 for every value of variances. ■

Considering this property, first sort tasks in terms of their target frequencies in each cluster. Then, proceed the following iteration in bother cluster and check whether capacity constraint can be satisfied.

- Select tasks that have the minimum target frequency.
- Check capacity constraint if the target frequency is increased to the next smallest target frequency.
- If capacity constraint is satisfied, then calculate the new target frequency that maximizes the total utilization, i.e., $\sum u^{big} = m^{big}$ or $\sum u^{LITTLE} = m^{LITTLE}$ in terms of the selected cluster and assign it to all selected tasks and end the iteration.
- If capacity constraint is not satisfied, then set the target frequency as same as the next smallest target frequency
- If all tasks are selected at the first statement, then calculate the minimum target frequency that meet the capacity constraint, i.e. $\sum u^{big} = m^{big}$ or $\sum u^{LITTLE} = m^{LITTLE}$ in terms of the selected cluster directly. This frequency may not be over the maximum frequency of the selected cluster. If the calculated frequency is larger than the maximum frequency, set the target frequency as the maximum frequency with broken capacity constraint.

*Example 1:* let there are four tasks in a task set with the target frequencies (500, 800, 1000, 1400) and assume that this cluster is not feasible. In the first iteration, the task whose target frequency 500 is selected and check the feasibility if the target frequency is increased to the next smallest frequency, 800. Assume that it still is not feasible. Then, this task now has the new target frequency 800, which was the next smallest frequency and the second iteration starts.

Note that when we complete the iteration completely, the number of tasks that have the minimum target frequency is not one. Then the first statement in the next iteration will select all those tasks. At the first statement in the second iteration, two tasks are selected because they have the same target frequency and is the smallest. Check again the feasibility is satisfied if the target frequency is increased to the next smallest frequency, 1000. Now assume that it becomes feasible. Then, the target frequency that maximizes the total utilization would be some point between 800 and 1000. Calculate this target frequency and set it to two tasks.

Suppose that every tasks have the target frequency 1400. In this situation, there is no next smallest target frequency. Then, by following the fifth statement, we directly calculate the minimum target frequency meeting the capacity constraint. This frequency value should be larger than 1400. ■

### 4) *Considering task migration*

After performing DVFS, it still cannot satisfy the capacity constraint. At the fifth statement in DVFS procedure, if the calculated target frequency is larger than the maximum frequency, the target frequency of all tasks is just set to the maximum frequency. It means the total amount of workload is larger than the total capacity of the cluster, so some task should be migrated to the other cluster. As the final assignment step, the following iteration makes the task set feasible.

- Sort tasks in a cluster that the capacity constraint is broken in terms of their big.LITTLE execution ratio $r_i$. If the cluster is the LITTLE cluster, sort by descending order. If it is the big cluster, sort by ascending order.
- Calculate whether feasibility can be satisfied when the first task in the sorted tasks is migrated. If it is still not feasible, migrate it to the other cluster and do the first statement again.
- If it is feasible at the second statement, fractionize the task and migrate a portion of the task to maximize the total utilization which this task is in, i.e., $\sum u^{big} = m^{big}$ or $\sum u^{LITTLE} = m^{LITTLE}$ in terms of the cluster. After fractionizing, increase target frequencies of tasks in the other cluster by performing DVFS procedure once again.
- If both capacity constraints are broken during iteration, it is not feasible.

Note that at the third statement, all tasks in the cluster, called cluster A, has the maximum frequency of the cluster with total utilization in the cluster close to one. However, it is still possible to contradict either capacity constraint or frequency constraint. During DVFS procedure in the other cluster, called cluster B, at the third statement, some adjusted target frequencies to meet its capacity constraint may be over than the maximum frequency of the cluster B, it violates the frequency constraint. To solve this problem, some portion of the task should be migrated once again to the cluster A but it cannot be migrated because the cluster A already has the total utilization 1 with the maximum frequency for all tasks by the third statement. As a result, it is not feasible.

*Example 2:* let one task in the LITTLE cluster should be migrated to the big cluster to be feasible. Then, the remaining two tasks have appropriate target frequency to maximize the total utilization in the LITTLE cluster, not the maximum frequency. This target frequency is less than or equal to the maximum frequency. ■

By summarizing all procedures, our VFS-Hetero-Split workload assignment algorithm is as follows.

**Algorithm 1. VFS-Hetero-Split Workload Assignment**

```
1:           1)   Considering deadline constraint
2:   foreach $\tau_i \in \tau$
3:      calculate $f_{target,i}^{big}$ and $f_{target,i}^{LITTLE}$
4:      if $f_{target,i}^{big} > f_{max}^{big}$ then
5:         return not feasible
6:      end if
7:           2)   Considering up-threshold
8:      if $f_{target,i}^{LITTLE} > f_{up,i}$ or $f_{target,i}^{LITTLE} > f_{max}^{LITTLE}$ then
9:         assign $\tau_i$ into $\pi^{big}$
10:     else
11:        assign $\tau_i$ into $\pi^{LITTLE}$
12:     end if
13:  end foreach
14:          3)   Considering DVFS
15:  if C3' is violated then
16:     sort tasks in $\pi^{big}$ in terms of $f_{target,i}$
17:     repeat
18:        $\{\tau\} \leftarrow$ the tasks with the minimum $f_{target}$
19:        if every tasks are selected then
20:           calculate the target frequency meeting C3'
21:           end repeat
22:        else if feasible when $f_{target,k}^{big} \leftarrow f_{target,k+1}^{big}$ then
23:           set the new $f_{target,\tau}$ to $\{\tau\}$ to make $\sum u^{big} = m^{big}$
24:           end repeat
25:        else
26:           $f_{target,k}^{big} \leftarrow f_{target,k+1}^{big}$
27:        end if
28:     until C3' is satisfied
29:  if C4' is violated then
30:     sort task in $\pi^{LITTLE}$ in terms of $f_{target,i}$
31:     do the corresponding jobs to lines 17-28.
32:  end if
33:          4)   Considering task migration
34:  if $f_{target,\tau}^{big} > f_{max}^{big}$ then
35:     sort tasks in $\pi^{big}$ in terms of $r_i$
36:     repeat
37:        $\tau \leftarrow$ the tasks with the minimum $r_i$
38:        if feasible when $\tau \rightarrow \pi^{LITTLE}$ then
39:           calculate $\tau^{big}$, $\tau^{LITTLE}$ of $\tau$ to make $\sum u^{big} = m^{big}$
40:           migrate $\tau^{LITTLE}$ into $\pi^{LITTLE}$
41:           do DVFS once again in lines 30-31.
42:           end repeat
43:        end if
44:     for every $\tau \in \pi^{big}$
45:  else if $f_{target,\tau}^{LITTLE} > f_{max}^{LITTLE}$ then
46:     sort tasks in $\pi^{LITTLE}$ in terms of $r_i$
47:     repeat
48:        $\tau \leftarrow$ the tasks with the maximum $r_i$
49:        if feasible when $\tau \rightarrow \pi^{big}$ then
50:           calculate $\tau^{big}$, $\tau^{LITTLE}$ of $\tau$ to make $\sum u^{LITTLE} = m^{LITTLE}$
51:           migrate $\tau^{big}$ into $\pi^{big}$
52:           do DVFS once again in lines 15-28
53:           end repeat
54:        end if
55:     for every $\tau \in \pi^{LITTLE}$
56:  end if
57:  if C3' or C4' is still violated then
58:     return not feasible
59:  else
60:     return $\tau^{big} \in \pi^{big}$, $\tau^{LITTLE} \in \pi^{LITTLE}$
61:  end if
```

## C. Properties of VFS-Hetero-Split

### 1) Requirements to be used with Hetero-Wrap algorithm

When introducing our system model, we said that we will use the existing Hetero-Fair guideline and Hetero-Wrap algorithm. To use these, workload assignments that are generated by VFS-Hetero-Split must have properties that are in workload assignments generated by the existing Hetero-Split algorithm. The original Hetero-Split algorithm guarantees the following two key properties:

- Among fractionally assigned tasks, there exists at most one task that has $u_i^1 + u_i^2 < 1$. The other tasks have $u_i^1 + u_i^2 = 1$.
- The number of tasks that are fractionally assigned to both type-1 and type-2 clusters is at most $m_1 + m_2$.

We will prove VFS-Hetero-Split algorithm also guarantees those properties.

*Proof:* In VFS-Hetero-Split algorithm, all constraints are satisfied via frequency scaling and total task migration, not workload fraction. Therefore, there is no fractionally assigned tasks except one task. In the task migration iteration, the last step of algorithm, only the last selected task is fractionized and has utilization $u^{big} + u^{LITTLE} \leq 1$. In the original Hetero-Split algorithm, $m_k$ is regarded as not only the total capacity of the type-k cluster, but also the number of cores in type-k cluster. Therefore, $m_1 + m_2$ cannot be less than 2. In VFS-Hetero-Split algorithm, there is at most one fractionized task, the second property is also always satisfied. Therefore, VFS-Hetero-Split algorithm guarantees the two properties. ■

Since every workload assignment created by VFS-Hetero-Split algorithm satisfies the above two properties, we can assign the task set into the cluster by using the existing Hetero-Wrap algorithm.

### 2) Time complexity

VFS-Hetero-Split algorithm consists of four procedures. In Algorithm 1, 1) and 2) requires $O(n)$ to calculate the target frequencies for big cluster and LITTLE cluster and some comparisons where $n$ is the number of tasks. In 3), we need a sorting mechanism which requires $O(nlogn)$. The repeating procedures only require $O(n)$, it does not affect time complexity. In 4), we sort tasks once again which requires $O(nlogn)$ and perform repetitions with $O(n)$, hence total $O(nlogn)$. For considering all four procedures, time complexity of Algorithm 1 is still at most $O(nlogn)$ which is same with the existing Hetero-Split algorithm.

## IV. Evaluation

In this section, we will measure how our algorithm works well, how energy consumption is reduced compared to calculated power consumption when cores are always running with their maximum frequencies.

## *A. Simulation environment*

As we don't have any power-measurable environment, we assumed the parameters to make power consumption as similar as possible to measured power consumption of Samsumg Exynos 7420 as follows:

TABLE II. THE PARAMETERS ABOUT POWER CONSUMPTION

| | **A53 LITTLE cluster** | **A57 big cluster** |
|---|---|---|
| Cuncore | 0 | 0 |
| Cstatic | 0.02 | 0.05 |
| Cdynamic | 0.13 | 0.39 |

A DVFS table that we used is from the kernel for Samsung Galaxy S6 which used Samsung Exynos 7420 processor. The DVFS table from kernel source is as follows:

TABLE III. THE DVFS TABLE BASED ON SAMSUNG EXYNOS 7420

| **A53 LITTLE cluster** | | **A57 big cluster** | |
|---|---|---|---|
| ***Voltage (mV)*** | ***Frequency (Ghz)*** | ***Voltage (mV)*** | ***Frequency (Ghz)*** |
| 675 | 0.4 | 900 | 0.8 |
| 712.5 | 0.5 | 925 | 0.9 |
| 750 | 0.6 | 950 | 1.0 |
| 787.5 | 0.7 | 981.25 | 1.1 |
| 825 | 0.8 | 1012.5 | 1.2 |
| 862.5 | 0.9 | 1043.75 | 1.3 |
| 900 | 1.0 | 1081.25 | 1.4 |
| 943.75 | 1.1 | 1118.75 | 1.5 |
| 981.25 | 1.2 | 1156.25 | 1.6 |
| 1018.75 | 1.3 | 1200 | 1.7 |
| 1068.75 | 1.4 | 1250 | 1.8 |
| 1118.75 | 1.5 | 1250 | 1.9 |
| - | - | 1250 | 2.0 |
| - | - | 1250 | 2.1 |

As Samsung Exynos 7420 has 4 big cores and 4 LITTLE cores, we also assumed $m^{big} = 4$ and $m^{LITTLE} = 4$. There is a parameter $r$ that represents the speed of the other devices except CPU. We assumed that this value is 0.4, which is the same speed with the least speed of a LITTLE core.

We generate 100 feasible task sets and compare how power consumption is reduced compared to when it is running with the maximum frequency of each cluster.

## *B. Experiment result*

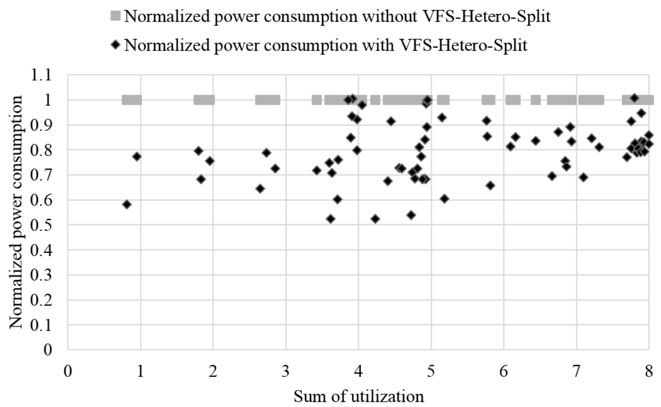

Fig. 4. Normalized power consumption assigned by VFS-Hetero-Split

Figure 4 shows that how power consumption can be reduced by VFS-Hetero-Split algorithm comparing running with the maximum frequency. Calculating the least target frequency by VFS-Hetero-Split algorithm, we could reduce power consumption up to 48% without losing any feasibility.

We observed that even though by using VFS-Hetero-Split, power consumption would be increased. This is because that the actual DVFS table is different from our assumption, the core voltage is proportional to the core frequency. In table 3, the core voltage of the big core is fixed to 1.25V when the frequency is larger than 1.8Ghz. Therefore, the power consumption rather can be decreased when the target frequency of the big core is larger than 1.8Ghz because the execution time would be decreased. In Figure 4, this phenomenon is observed when sum of power utilization is around 4 and 8. It means that even with consideration on task migration, the target frequency of the big core approaches to the maximum frequency, so normalized power consumption approaches to one or becomes larger than one. This phenomenon can be shown more clear if we show power consumption on each cluster individually:

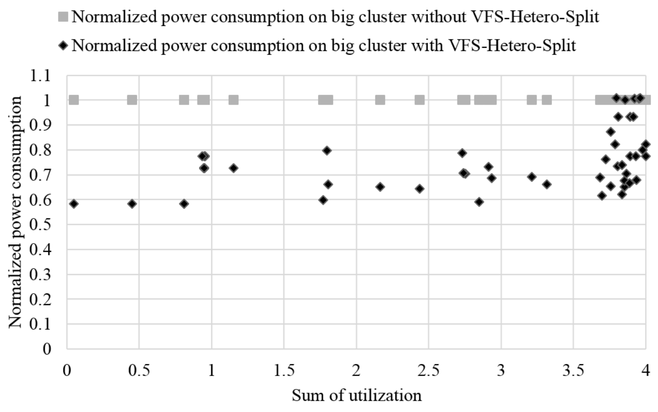

Fig. 5. Normalized power consumption on big cluster

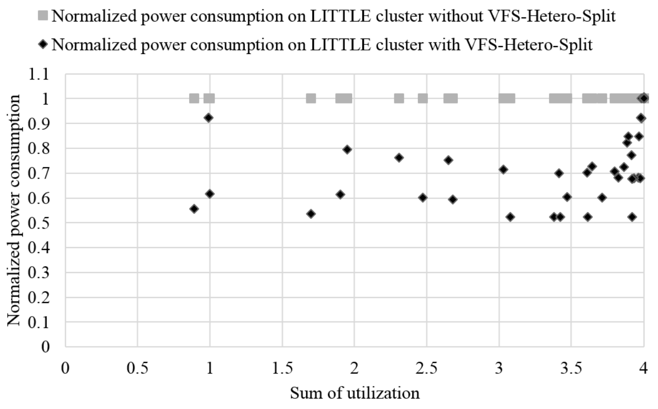

Fig. 6. Normalized power consumption on LITTLE cluster

In Figure 5, there are much more points with normalized power consumption around one in the area that sum of utilization is larger than 3.5, whereas there are no points that approaches to the line of normalized power consumption becomes one except sum of utilization becomes exactly four in Figure 6.

## V. Conclusion

We implemented the workload assignment algorithm, VFS-Hetero-Split, considering several properties of heterogeneous multicore processing platform. Our algorithm is an extended version of Hetero-Split algorithm and also guarantees the properties of Hetero-Split, every workload assignment generated by VFS-Hetero-Split can be assigned by Hetero-Fair guideline and Hetero-Wrap algorithm.

Our purpose is to decrease power consumption to complete a task set without losing feasibility. As power consumption is proportional to running frequency, we should run tasks with the minimum frequency. To achieve this goal, our algorithm assigns tasks based on the following characteristics of HMP architecture: workload decomposition, up-threshold, DVFS, and task migration.

As we can freely adjust the execution time and running frequency, considering off-cpu workload significantly affects calculation of target running frequency. Because tasks have different properties such as the number of branches and parallel execution degree, they may have different values of execution ratio. After calculating the initial target frequency for both clusters, we consider running the task on which cluster is more energy efficient by using up-threshold. We approximately calculate the value of up-threshold and compare it with the target frequency of the LITTLE cluster and assign it onto appropriate cluster. After assigning tasks, we adjust the target frequency by DVFS to satisfy capacity constraint. During this procedure, frequency constraint can be violated. To solve this problem, we migrate entire or some portion of tasks to the other cluster.

We generated feasible task sets and tested how power consumption can be reduced when we use VFS-Hetero-Split. We showed that our algorithm can reduce power consumption up to 48% comparing that when the tasks is running with the maximum frequency, without losing any feasibility.

## VI. Related Works

The optimization of HMP in real-time system is based on the different characteristics between architecture big and LITTLE such as performance and power efficiency. However, the study covering optimal real-time scheduling on two-type heterogeneous multicore platform suggested the fully-migrative scheduling framework [1]. By combining Hetero-Fair and Hetero-Split, they developed the first optimal two-type heterogeneous multicore scheduling algorithm. However, they have some limitation that they did not consider DVFS widely used in practice. Some studies tried to model heterogeneous multicore platform by combining two cluster of CPU [8]. However, since the type of two cluster is same, it is reasonable to call it as asymmetric multicore instead of heterogeneous multicore. Previous study to consider both energy and feasibility at the same time successfully modeled on ARM big.LITTLE heterogeneous multicore platform [6]. However, since it implemented the models as only for cluster switching, not HMP, it is less power efficient in terms of energy consumption. Another study covering power efficiency and CPU governing in heterogeneous multicore platform verified the power consumption model of the AP [7]. Based on the power consumption model, they suggested Medusa which is a new philosophy to control CPU frequency and they found Medusa can reduce the power consumption without any penalty of the performance. But, their assumption that the feasibility is always guaranteed is not fairly practical. Also, since the power curve of Samsung Exynos 5410 that they used is totally different from that of current HMP architectures such as Samsung Exynos 7420, it is becoming less useful as time goes on.

## References

[1] Hoonsung Chwa, and Jaebaek Seo, “Optimal Real-Time Scheduling on Two-Type Heterogeneous Multicore Platforms”, *IEEE Real-Time Systems Symposium (RTSS 2015)*, 2015

[2] Hongsuk Chung, Munsik Kang, and Hyunduk Cho, “Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM big.LITTLE Technology”, Samsung Electronics, 2013. [Online]. Available: https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf

[3] Brian Jeff, “big.LITTLE Technology Moves Towards Fully Heterogeneous Global Task Scheduling”, ARM, 2013. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_technology_moves_towards_fully_heterogeneous_Global_Task_Scheduling.pdf

[4] Andrei Frumusanu and Ryan Smith, “ARM A53/A57/T760 investigated – Samsung Galaxy Note 4 Exynos Review”, Anandtect, 2015. [Online]. Available: http://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/5

[5] Andrei Frumusanu, “The Samsung Exynos 7420 Deep Dive – Inside A Modern 14nm SoC”, Anandtech, 2015. [Online]. Available: http://www.anandtech.com/show/9330/exynos-7420-deep-dive

[6] Hoonsung Chwa and Jaebaek Seo, “Energy and Feasibility Optimal Global Scheduling Framework on big.LITTLE platforms”, in *RTSOPS* July 7 2015

[7] Aaron Carroll and Gernot Heiser, “Unifying DVFS and Offlining in Mobile Multicores”, in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014 IEEE 20th

[8] Sungil Kim, Hwantae Kim, “Using DVFS and Task Scheduling Algorithms for a Hard Real-Time Heterogeneous Multicore Processor Environment” In *Proceedings of the 2013 workshop on Energy efficient high performance parallel and distributed computing*, June 17 2013

[9] Samsung Electronics, “SM-G920S_LL_Opensource”, 2015. [Online]. Available: http://opensource.samsung.com/reception.do

[10] Linda Null and Julia Lobur, “Amdahl’s Law”, in *The Essentials of Computer Organization and Architecture, 3rd Ed*, MA: Jones & Bartlett, 2010, pp 402-405

[11] Kihwan Choi and Massoud Pedram, “Dynamic Voltage and Frequency Scaling for Energy-Efficient System Design”, University of Southern California, April 27 2005

[12] Mostafa Abd-El-Barr and Hesham El-Rewini, “Memory Hierarchy”, in *Fundamentals of Computer Organization and Architecture*, New York: Wiley, 2013, pp 107-109

[13] S. Baruah, “Task partitioning upon heterogeneous multiprocessor platforms”, in *RTAS*, 2004

[14] Android Open Source Project, “Scheduling algorithm for HMP”, 2014. [Online]. Available: https://github.com/dtsinc/DTS-Eagle-Integration_CAF-Android-kernel/blob/master/Documentation/scheduler/sched-hmp.txt