Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment

Xue Lin, *Student Member, IEEE,* Yanzhi Wang, *Student Member, IEEE,* Qing Xie, *Student Member, IEEE,* and Massoud Pedram, *Fellow, IEEE*

Abstract—Mobile cloud computing (MCC) offers significant opportunities in performance enhancement and energy saving for mobile, battery-powered devices. Applications running on mobile devices may be represented by task graphs. This work investigates the problem of scheduling tasks (which belong to the same or possibly different applications) in the MCC environment. More precisely, the scheduling problem involves the following steps: (i) determining the tasks to be offloaded onto the cloud, (ii) mapping the remaining tasks onto (potentially heterogeneous) local cores in the mobile device, (iii) determining the frequencies for executing local tasks, and (iv) scheduling tasks on the cores (for in-house tasks) and the wireless communication channels (for offloaded tasks) such that the task-precedence requirements and the application completion time constraint are satisfied while the total energy dissipation in the mobile device is minimized. A novel algorithm is presented, which starts from a minimal-delay scheduling solution and subsequently performs energy reduction by migrating tasks among the local cores and the cloud and by applying the dynamic voltage and frequency scaling technique. A linear-time rescheduling algorithm is proposed for the task migration. Simulation results demonstrate significant energy reduction with the application completion time constraint satisfied.

Index Terms—mobile cloud computing (MCC), task scheduling, DVFS, energy minimization, hard deadline constraint

1 Introduction

Mobile devices e.g., smart-phones and tablet-PCs, have been widely employed as a major computing platform due to their portability and compactness. However, the increase of the volumetric/gravimetric energy density of rechargeable batteries falls behind the increase of the power demand of mobile devices with functionality improvements, thus, resulting in a shorter battery life for mobile devices and also a power crisis in the development of mobile device technology [2]. It is also noteworthy that mobile devices have relatively weak computing resources compared to their "wall-powered" counterparts due to the constraints on weight, size and power.

Cloud computing has been recognized as a prospective computing paradigm because of its potential benefits such as on-demand service, ubiquitous network access, location independent resource pooling, and transference of risk [3], [4], [5]. In the cloud computing paradigm, there are a service provider who owns and manages the centralized computing and storage resources in the cloud, and users who have access to those resources over the Internet.

- *X. Lin, Y. Wang, Q. Xie, and M. Pedram are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, 90089.*
  E-mail: {xuelin, yanzhiwa, xqing, pedram}@usc.edu

A preliminary version of this paper was presented at the 2014 IEEE International Conference on Cloud Computing (CLOUD), Anchorage, AK, USA [1].

With the development of wireless communication technology such as 3G, Wi-Fi, and 4G, the mobile cloud computing (MCC) paradigm has emerged to shift the processing, memory, and storage requirements from the resource-limited mobile devices to the resource-unlimited cloud computing system [6], [7], [8]. MCC can improve the performance of mobile devices by (i) selectively offloading tasks of an application (e.g., object/gesture recognition, image/video editing, natural language processing, and scientific computation [9], [10]) onto the cloud and (ii) carefully scheduling local task executions in the mobile device with the anticipation of remote task executions in the cloud while taking into account the task-precedence requirements. Task offloading can help to improve the performance of mobile devices because servers in the cloud have much larger computation capability and higher speed than the mobile processor. Moreover, MCC helps to save energy in mobile devices and prolong operation time of the battery by offloading computation-intensive tasks onto the cloud. Experiments conducted in [11], [12] demonstrate that (i) a large application can be partitioned into various tasks with task-precedence requirements, and (ii) the fine granularity of task-level offloading has the potential to achieve lower energy consumption and higher performance.

In order to realize the prospective benefits of MCC in energy saving and performance improvement for mobile devices, we should consider the following

questions. (i) Which tasks of an application should be offloaded onto the cloud? (ii) How to map the remaining local tasks onto the potentially heterogeneous cores in a mobile device? (iii) Which execution frequency level should be assigned for each local task? (iv) How to schedule tasks in the heterogenous cores for in-house processing and in the wireless communication channels for remote processing, such that the task-precedence requirements and application completion time constraint are satisfied with the minimum energy consumption in the mobile device? (Please note that although the mobile device cannot schedule task executions in the cloud, it can anticipate and estimate the execution time of offloaded tasks based on its prior knowledge.)

To answer the above-mentioned questions, this work differs from previous work by focusing on the *MCC task scheduling* problem, in which there are four key issues to be addressed.

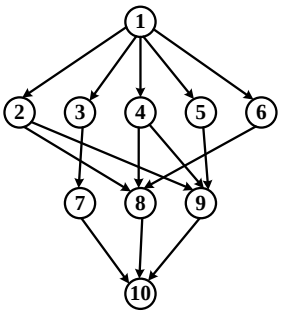
- The application completion time constraint is a hard constraint, and therefore it should be addressed in the first place. Offloading computation-intensive tasks onto the cloud may result in less application completion time. However, the offloading decision should be made judiciously considering the delay due to uploading/downloading data to/from the cloud.
- The total energy consumption in a mobile device for executing an application, including the energy consumed both by the processing units (i.e., the potentially heterogeneous cores in the mobile device) and by the RF components for offloading tasks, is the objective function to be minimized. From the perspective of energy consumption, offloading tasks onto the cloud saves the computation energy but induces the communication energy.
- The dynamic voltage and frequency scaling (DVFS) technique can be employed for further reducing energy consumption of a mobile device. However, if the DVFS technique is applied (to lower the execution frequency of a high performance core) such that a high performance core has the same performance as a low performance core, the high performance core still consumes more energy than the low performance core [22]. Therefore, DVFS should be applied after determining the task assignment (to local cores and to the cloud).
- The task-precedence requirements should be enforced during task scheduling. Unlike the conventional local task scheduling problem in [13], in the MCC task scheduling problem there exist additional task-precedence requirements through wireless communication channels between the cloud and the local cores.

In this present work, we propose a novel algorithm for the MCC task scheduling problem to minimize the total energy consumption of an application in a mobile device under a hard constraint on the application completion time. In particular, we generate a minimal-delay task scheduling in the first step, and then perform energy reduction in the second step by migrating tasks towards the cloud or other local cores that can bring great energy reduction without violation of the application completion time constraint. In the third step, we apply the DVFS technique to further reduce energy consumption. To avoid high time complexity, we propose a linear-time rescheduling algorithm for the task migrations. In summary, the proposed novel algorithm can generate a task schedule at the beginning of application execution to minimize the total energy consumption under a hard application completion time constraint. The simulation results show that the proposed algorithm can achieve a maximum energy reduction of 74.9% compared with the baseline algorithms.

To our best knowledge, this is the first task scheduling work that minimizes energy consumption under a hard completion time constraint for the task graph in the MCC environment, taking into account the joint task scheduling on the local cores and the wireless communication channels of the mobile devices as well as on the cloud.

2 Related Work

Task scheduling and task offloading problems have been extensively studied and various heuristic algorithms have been proposed [13]∼[20]. These work can be classified into two categories: (i) minimizing the total application completion time (i.e., achieving higher performance) [13], [14], [15], [16] and (ii) minimizing the overall energy consumption (i.e., achieving longer battery life of battery-powered mobile devices) [17], [18], [19], [20]. The HEFT algorithm in [13] was proposed for scheduling tasks of an application with task-precedence requirements on heterogeneous processors with the objective of achieving high performance. This algorithm computes priorities of all tasks, selects a task with the highest priority value at each step, and assigns the selected task to the processor that minimizes the task's finish time. Alternatively, the Push-Pull algorithm starts from a fast deterministic task scheduling algorithm and then iteratively improves the current solution by using a deterministic guided search method [14]. Ra et al. [15] adopted an incremental greedy strategy and developed a runtime system, which is able to adaptively make offloading and parallel execution decisions for mobile interactive perceptual applications in order to minimize the completion time of applications. A genetic algorithm was proposed in [16] to optimize the partitioning of tasks of a data stream application between a mobile device and the cloud for the maximum throughput.



Task	Exe. Time Core1	Exe. Time Core2	Exe. Time Core3
1	9	7	5
2	8	6	5
3	6	5	4
4	7	5	3
5	5	4	2
6	7	6	4
7	8	5	3
8	6	4	2
9	5	3	2
10	7	4	2

$$1 \le \boldsymbol{i} \le \boldsymbol{N}, \begin{cases} \boldsymbol{T_i^s} = 3 \\ \boldsymbol{T_i^c} = 1 \\ \boldsymbol{T_i^r} = 1 \end{cases}$$

Fig. 1. A simplified example of a task graph.

Rong et al. [17] addressed the problem of minimizing energy consumption of a computer system executing periodic tasks, assuming that the periods of tasks are large enough such that the positive slack time between tasks can be used for energy consumption reduction. Li et al. [18] formulated the task mapping problem as a maximum-flow/minimum-cut problem to optimize the partitioning of a task graph between a mobile device and the cloud for the minimum energy consumption. Lee et al. [19] extended the work of [13] on heterogeneous processors accounting for both the energy consumption and application completion time. However, the algorithm in [19] cannot guarantee that the scheduling result meets a hard constraint of application completion time. Kumar et al. [20] proposed a straightforward offloading decision strategy to minimize the energy consumption according to the *computation-to-communication ratio* and the networking environment.

3 System Model for MCC Task Scheduling

3.1 Applications

An application is represented by a directed acyclic *task graph* $G = (V, E)$. Each node $v_i \in V$ represents a task and a directed edge $e(v_i, v_j) \in E$ represents the precedence constraint such that task (node) v_i should complete its execution before task (node) v_j starts execution. There are a total number of N tasks (nodes) in the task graph (application). Normally, N is smaller than 100, e.g., in computer vision applications, N is in the range of 10∼30 [15]. Given a task graph, the task without any parent is called the *entry task*, and the task without any child is called the *exit task*. In the task graph of an application, there may exist multiple entry tasks and multiple exit tasks. As shown in Fig. 1, task v_1 is the entry task and task v_{10} is the exit task. For each task v_i, we define $data_i$ and $data_i'$ as the number of bits of the task specification plus input data required for task uploading and the number of bits needed to download result and any other output data from the cloud after executing task v_i in the cloud, respectively.

3.2 MCC Environment

We consider a mobile device in the MCC environment that has access to the computing resources on the cloud. There are a number of K heterogeneous cores in the processor of the mobile device. Examples are the state-of-the-art Tegra [21] and big.LITTLE architecture [22] which has been adopted by Broadcom, Samsung, etc. The cores are equipped with a DVFS capability such that each core can operate at M different frequency levels (and the corresponding M supply voltage levels.) The maximum operating frequency of the k-th core is f_k^{max}, and there are M frequency scaling factors i.e., $a_{k,1} < a_{k,2} < ... < a_{k,M} = 1$. Therefore, the actual operating frequency of the k-th core can be $f_k = a_{k,m} \cdot f_k^{max}$. The (average) power consumption P_k of the k-th core is a super-linear function of f_k, represented by $P_k = \alpha_k \cdot (f_k)^{\gamma_k}$, where $2 \le \gamma_k \le 3$. The α_k and γ_k values may be different for different heterogeneous cores.

A task can be executed either locally on a core of the mobile device or remotely on the cloud. If task v_i is offloaded onto the cloud, there are three phases in sequence associated with the execution of task v_i: (i) the *RF sending* phase, (ii) the *cloud computing* phase, and (iii) the *RF receiving* phase. In the RF sending phase, the specification and input data of task v_i are sent to the cloud by the mobile device through the *wireless sending channel*. In the cloud computing phase, task v_i is executed in the cloud. In the RF receiving phase, the mobile device receives the output data of task v_i from the cloud through the *wireless receiving channel*. The cloud transmits the output data of task v_i back to the mobile device as long as it finishes processing task v_i. We use R^s to denote the data sending rate of the wireless sending channel, and R^r to denote the data receiving rate of the wireless receiving channel. Accordingly, let P^s denote the power consumption level of the RF component in the mobile device for sending data to the cloud. The power consumption of the RF component in the mobile device for receiving data is negligible compared to that for sending data.

The local core in the mobile device or the wireless sending channel can only process or send one task at a time, and preemption is not allowed in this framework. On the other hand, the cloud can execute a large number of tasks in parallel as long as there is no task-precedence requirements among the tasks.

3.3 Task-Precedence Requirements in the MCC Environment

We use $T_{i,k}^{l,min}$ to denote the minimum execution time of task v_i on the k-th core when the maximum

operating frequency f_k^{max} is used, where superscript l means "local execution". $T_{i,k}^{l,min}$ depends on the task specification of task v_i and the maximum operating frequency f_k^{max} of the k-th local core. Usually, $T_{i,k}^{l,min}$ is a decreasing function of f_k^{max}. Then, the actual execution time $T_{i,k}^{l}$ of task v_i on the k-th core depends on the actual operating frequency $f_k = a_{k,m} \cdot f_k^{max}$ by

$$T_{i,k}^{l} = T_{i,k}^{l,min}/a_{k,m}. \quad (1)$$

We use T_i^c to denote the computation time of task v_i on the cloud, where superscript c means "execution on the cloud". The time for sending task v_i onto the cloud, denoted by T_i^s, is calculated as:

$$T_i^s = data_i/R^s. \quad (2)$$

The time for receiving task v_i from the cloud, denoted by T_i^r, is calculated as:

$$T_i^r = data_i'/R^r. \quad (3)$$

For a task v_j that is already scheduled (on a local core or the cloud), we use FT_j^l, FT_j^{ws}, FT_j^c, and FT_j^{wr} to denote the finish times of task v_j on a local core, the wireless sending channel (i.e., the task has been completely offloaded to cloud), the cloud, and the wireless receiving channel (i.e., the mobile device has completely received the output data of the task from the cloud), respectively. If the task v_j is scheduled locally, we set $FT_j^{ws} = FT_j^c = FT_j^{wr} = 0$; if the task v_j is offloaded onto the cloud, we set $FT_j^l = 0$. Please note that the mobile device can only schedule tasks in the local cores and the wireless channels, whereas the cloud computing controller schedules tasks that have already been uploaded to the cloud and transmits the output data back to the mobile device. However, the mobile device can anticipate the execution of tasks in the cloud and estimate the corresponding FT_j^c and FT_j^{wr} values from the parameters T_j^c, T_j^r, etc.

3.3.1 Local Scheduling

Before we schedule a task v_i, all its immediate predecessors must have already been scheduled. Suppose that task v_i is to be scheduled on a local core. Then the *ready time* of task v_i, denoted by RT_i^l, is calculated as:

$$RT_i^l = \max_{v_j \in \mathbf{pred}(v_i)} \max\{FT_j^l, FT_j^{wr}\}, \quad (4)$$

where $\mathbf{pred}(v_i)$ is the set of immediate predecessors of task v_i. The ready time RT_i^l is the earliest time when all immediate predecessors of task v_i have completed execution and their results are available to task v_i:

- If task v_j (an immediate predecessor of task v_i) has been scheduled locally, $\max\{FT_j^l, FT_j^{wr}\} = FT_j^l$. In this case we have $RT_i^l \geq FT_j^l$, which means that task v_i can start execution on a local core only after the local execution of task v_j has finished.
- If task v_j (an immediate predecessor of task v_i) has been offloaded onto the cloud, $\max\{FT_j^l, FT_j^{wr}\} = FT_j^{wr}$. In this case we have $RT_i^l \geq FT_j^{wr}$, which means that task v_i can start execution on a local core only after the mobile device has completely received the output data (results) of task v_j through the wireless receiving channel.

We can only schedule task v_i to start execution at or after its ready time RT_i^l, if the task is to be scheduled on a local core. In this way the task-precedence requirements can be preserved. However, the mobile device might not be able to start executing task v_i at time RT_i^l exactly, because the cores may be executing other tasks at time RT_i^l.

3.3.2 Cloud Scheduling

On the other hand, suppose that task v_i is to be offloaded onto the cloud. The ready time of task v_i on the wireless sending channel, denoted by RT_i^{ws}, is calculated as:

$$RT_i^{ws} = \max_{v_j \in \mathbf{pred}(v_i)} \max\{FT_j^l, FT_j^{ws}\}. \quad (5)$$

RT_i^{ws} denotes the earliest start time when task v_i can be scheduled on the wireless sending channel in order to preserve the task-precedence requirements:

- If task v_j (an immediate predecessor of task v_i) has been scheduled locally, $\max\{FT_j^l, FT_j^{ws}\} = FT_j^l$. In this case we have $RT_i^{ws} \geq FT_j^l$, which means that the mobile device can start to send task v_i through the wireless channel only after the local execution of task v_j has finished.
- If task v_j (an immediate predecessor of task v_i) has been offloaded onto the cloud, $\max\{FT_j^l, FT_j^{ws}\} = FT_j^{ws}$. In this case we have $RT_i^{ws} \geq FT_j^{ws}$, which means that the mobile device can start to send task v_i through the wireless channel only after the mobile device has completed offloading task v_j to the cloud.

The ready time of task v_i on the cloud, denoted by RT_i^c, is calculated as:

$$RT_i^c = \max\{FT_i^{ws}, \max_{v_j \in \mathbf{pred}(v_i)} FT_j^c\}. \quad (6)$$

RT_i^c denotes the earliest time when task v_i can start execution on the cloud. If task v_j (an immediate predecessor of task v_i) is scheduled locally, $FT_j^c = 0$. Therefore, $\max_{v_j \in \mathbf{pred}(v_i)} FT_j^c$ in (6) is the time when all the immediate predecessors of task v_i that are offloaded to the cloud have finished execution on the cloud. On the other hand, FT_i^{ws} is the time when task v_i has been completely offloaded to the cloud through the wireless sending channel, and therefore we have $RT_i^c \geq FT_i^{ws}$. The cloud computing controller can schedule task v_i to start execution at time RT_i^c exactly (because of the high parallelism in the cloud),

such that the task-precedence requirements can be preserved.

Finally, let RT_i^{wr} denote the ready time for the cloud to transmit back the results of task v_i, and we have:

$$RT_i^{wr} = FT_i^c. \tag{7}$$

In other words, the cloud can transmit the output data (results) of task v_i back to the mobile device immediately after it has finished processing this task.

3.4 Energy Consumption and Application Completion Time

If task v_i is executed locally on the k-th core of the mobile device, the energy consumption of the task is given by:

$$\begin{aligned} E_{i,k}^l &= P_k \cdot T_{i,k}^l \\ &= \alpha_k \cdot (f_k)^{\gamma_k} \cdot T_{i,k}^{l,min}/a_{k,m} \\ &= \alpha_k \cdot (a_{k,m} \cdot f_k^{max})^{\gamma_k} \cdot T_{i,k}^{l,min}/a_{k,m} \\ &= (a_{k,m})^{(\gamma_k - 1)} \cdot \alpha_k \cdot (f_k^{max})^{\gamma_k} \cdot T_{i,k}^{l,min} \\ &= (a_{k,m})^{(\gamma_k - 1)} \cdot E_{i,k}^{l,max}, \end{aligned} \tag{8}$$

where $E_{i,k}^{l,max}$ is the energy consumption for executing task v_i on the k-th local core under the maximum operating frequency. Please note that both P_k and $T_{i,k}^l$ depend on the operating frequency of the k-th core. If task v_i is offloaded to the cloud, the energy consumption of the mobile device for offloading the task is given by:

$$E_i^c = P^s \cdot T_i^s. \tag{9}$$

The execution of task v_i on the cloud does not consume energy of the mobile device. The energy consumption of receiving the results of task v_i is negligible. The total energy consumption of the mobile device for running the application, denoted by E^{total}, is given by:

$$E^{total} = \sum_{i=1}^{N} E_i. \tag{10}$$

where E_i equals to $E_{i,k}^l$ if task v_i is executed locally on the k-th core of the mobile device, and equals to E_i^c if the task is offloaded to the cloud.

The application completion time T^{total} is calculated by:

$$T^{total} = \max_{v_i \in exit\ tasks} \max\{FT_i^l, FT_i^{wr}\}. \tag{11}$$

The inner max block gives the finish time of an exit task v_i. It equals to FT_i^l if task v_i is executed on a local core, and equals to FT_i^{wr} if task v_i is offloaded to the cloud.

The MCC task scheduling problem is to (i) determine the tasks of an application to be offloaded, (ii) map the remaining tasks onto heterogeneous cores in the mobile device, (iii) determine the execution frequency for each local task, and (iv) schedule the tasks on heterogeneous cores and wireless communication channels. The objective is to minimize E^{total} under the following constraints: (i) task-precedence requirements and (ii) the application completion time constraint $T^{total} \leq T^{max}$, where T^{max} is the maximum application completion time.

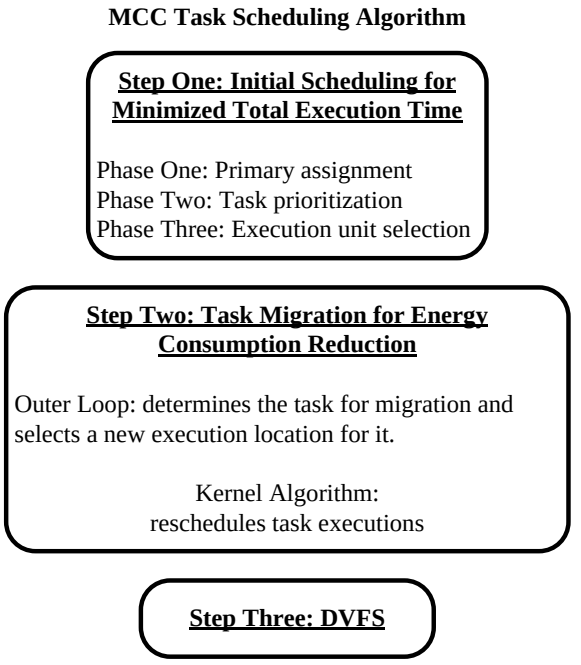


Fig. 2. Flow chart of the MCC task scheduling algorithm.

4 MCC Task Scheduling Algorithm

The MCC task scheduling algorithm has three steps: initial scheduling for minimizing the application completion time T^{total}, task migration for minimizing the energy consumption E^{total}, and DVFS for further reducing the energy consumption. In all the three steps, the task-precedence and application completion time constraints should be enforced. The flow chart of the whole MCC task scheduling algorithm is shown in Fig. 2.

In order to strictly satisfy the application completion time constraint, we minimize T^{total} in the first step and then reduce energy consumption by task migration and DVFS. Otherwise, if we minimize energy consumption at first, the application completion time constraint can hardly be guaranteed because of the task-precedence requirements and the parallelism constraints on the local cores and the wireless communication channels.

4.1 Step One: Initial Scheduling Algorithm

In the initial scheduling algorithm, we generate the minimal-delay schedule without considering the energy consumption of the mobile device. The HEFT

algorithm [13] generates the minimal-delay scheduling for tasks running on a number of heterogeneous cores. We modify the HEFT algorithm to take into account the joint scheduling of tasks on the local cores, the wireless communication channels, and the cloud. The initial scheduling algorithm has three phases: primary assignment, task prioritization, and execution unit selection, as shown in Fig. 2. In the following, we discuss the three phases in detail.

4.1.1 Primary Assignment

In this phase, we determine the subset of tasks that are initially assigned for the cloud execution. Offloading such tasks to the cloud will result in savings of the application completion time. Please note that this primary assignment is not the final decision, since we can assign more tasks for remote execution in the "execution unit selection" phase of initial scheduling. For each task v_i, we calculate the minimum local execution time $T_i^{l,min}$ as:

$$T_i^{l,min} = \min_{1 \le k \le K} T_{i,k}^{l,min}. \tag{12}$$

We also calculate the estimated remote execution time T_i^{re} as:

$$T_i^{re} = T_i^s + T_i^c + T_i^r. \tag{13}$$

If $T_i^{re} < T_i^{l,min}$, task v_i is assigned for remote execution on the cloud. We call such task a "primary cloud task".

4.1.2 Task Prioritization

In this phase, we calculate the *priority* of each task similar to the HEFT algorithm. First, we calculate the *computation cost* w_i for each task. If task v_i is a cloud task, its computation cost is given by

$$w_i = T_i^{re}. \tag{14}$$

If task v_i is not a cloud task, w_i is calculated as the average computation time of task v_i in the local cores, i.e.,

$$w_i = \operatorname*{avg}_{1 \le k \le K} T_{i,k}^{l,min}. \tag{15}$$

Please note that we assume the maximum operating frequency for each local core in the initial scheduling step and the task migration step. The priority level of each task v_i is recursively defined by

$$priority(v_i) = w_i + \max_{v_j \in \mathbf{succ}(v_i)} priority(v_j), \tag{16}$$

where $\mathbf{succ}(v_i)$ is the set of immediate successors of task v_i. The priority levels are recursively computed by traversing the task graph starting from the exit tasks. For the exit tasks, the priority level is equal to

$$priority(v_i) = w_i, v_i \in exit\ tasks. \tag{17}$$

Basically, $priority(v_i)$ is the length of the critical path from task v_i to the exit tasks.

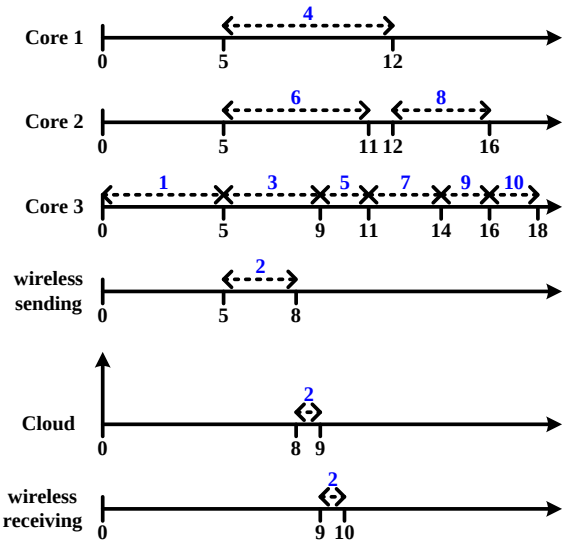


Fig. 3. Task scheduling result generated by the initial scheduling algorithm.

4.1.3 Execution Unit Selection

In this phase, tasks are selected and scheduled in the descending order of their priorities. If task v_j is the immediate predecessor of task v_i, we have $priority(v_j) > priority(v_i)$ from (16). Therefore, when task v_i is selected for scheduling in this phase, all its immediate predecessors have already been scheduled.

- If the selected task v_i is a primary cloud task, we calculate its ready time RT_i^{ws} on the wireless channel, and allocate the earliest available time slot on the wireless sending channel for offloading the task. Please note that the mobile device might not be able to start offloading task v_i at time RT_i^{ws} if it is offloading other tasks at that time. We calculate FT_i^{ws} accordingly, and then the cloud will begin executing task v_i at the ready time RT_i^c (from (6)) (because of the high parallelism in the cloud.) Finally we calculate $FT_i^c = RT_i^c + T_i^c$ and $FT_i^{wr} = FT_i^c + T_i^r$. In this way, we have scheduled task v_i and estimated the associated finish time.
- If the selected task v_i is not a primary cloud task, it may be scheduled on a local core or the cloud. We need to estimate the finish time of this task if it is scheduled on each core using the maximum operating frequency and the finish time of this task if it is offloaded to the cloud, using the similar procedure as described above. Then we schedule task v_i on the core or offload it to the cloud such that the finish time is minimized. When we schedule the task, we need to make sure that the task-precedence requirements are satisfied according to Section 3.3.

As an example, we perform initial task scheduling on the task graph shown in Fig. 1, assuming that there are three heterogeneous cores in the mobile device.

The $T_{i,k}^{l,min}$ values are shown in the table in Fig. 1, and in reality these values are determined by the characteristics of the tasks and the heterogenous cores. We use $T_i^s = 3$, $T_i^c = 1$, and $T_i^r = 1$ for all the tasks for simplicity, and in reality these values are set according to (2), (3), and the computation speed of the cloud servers. Fig. 3 presents the task scheduling result, where the horizontal axes denote the time. For example, task v_4 is executed on core 1 from time 5 to 12. Task v_2 is offloaded onto the cloud. The mobile device sends the specification and input data of task v_2 using the wireless sending channel from time 5 to 8. And then, task v_2 is computed on the cloud from time 8 to 9. The cloud transmits the output data (results) of task v_2 back to the mobile device from time 9 to 10. The application completion time of this example is 18, which is the finish time of the exit task v_{10}.

4.2 Step Two: Task Migration Algorithm

The task migration algorithm aims at minimizing the energy consumption E^{total} under the application completion time constraint $T^{total} \leq T^{max}$. The energy consumption is reduced through migrating tasks from a local core to another local core or to the cloud. The task migration algorithm is an iterative algorithm comprised of a *kernel algorithm* and an *outer loop* as shown in Fig. 2. In each iteration, the outer loop determines the target task for migration and the new execution location (i.e., a different local core or the cloud) in order to minimize the energy consumption E^{total}. It should also maintain the application time constraint $T^{total} \leq T^{max}$ without violation. Given the target task for migration and the new execution location, the kernel algorithm generates a new scheduling result that has the minimum application completion time T^{total} with linear time complexity.

4.2.1 Outer Loop

The outer loop of the task migration algorithm determines the target tasks to migrate from one local core to another local core or to the cloud, in order to reduce energy consumption of the mobile device. It should also maintain the application completion time constraint $T^{total} \leq T^{max}$ without violation. Please note that the task migration algorithm does not account for the migration of a task from offloading to the cloud back to local processing, because the energy consumption of the mobile device will generally increase in this case.

In each iteration of the outer loop, let N' denote the number of tasks that are currently scheduled on the local cores. Each of them can be moved to execute on one of the other $K-1$ cores or the cloud. Therefore, there are a total of $N' \times K$ migration choices.

- For each choice, we run the kernel algorithm to find a new schedule, and calculate the corresponding energy consumption E^{total} and application completion time T^{total}.
- We select the choice that results in the largest energy reduction compared with the current schedule and no increase in the application completion time T^{total} than the current schedule.
- If we cannot find such a choice, we select the one that results in the largest ratio of energy reduction to the increase of the application completion time. We should make sure that the new application completion time does not exceed the limit value T^{max}.

We repeat the previous steps until the energy consumption of the mobile device cannot be further minimized without violation of the application completion time constraint.

4.2.2 Kernel Algorithm (i.e., Rescheduling Algorithm)

In a task schedule, let k_i denote the execution location of task v_i. $k_i \neq 0$ means that task v_i is executed on the k_i-th core, whereas $k_i = 0$ means that task v_i is offloaded onto the cloud. In the kernel algorithm, we have had an original schedule of the task graph. We are given by the outer loop a task v_{tar} for migration and its new execution location k_{tar}. The kernel algorithm should generate a new schedule of the task graph G, where task v_{tar} is executed on the new location k_{tar} and the remaining tasks are executed on the same locations as in the original schedule. The kernel algorithm aims at minimizing the application completion time T^{total}. On the other hand, the energy consumption E^{total} is fixed and can be directly calculated using (8)$\sim$(10) once the execution locations of tasks are known. Because the kernel algorithm will be called many times from the outer loop, we propose an efficient linear-time rescheduling algorithm of the task graph as the kernel algorithm, which is more efficient than the modified HEFT algorithm when the number of cores is relatively large.

For the original schedule, we use a *sequence set* $S_k = \{v_{(k,1)}, v_{(k,2)}, \dots\}$ to denote the sequence of tasks that are executed on the k-th local core and we use the sequence set $S_0 = \{v_{(0,1)}, v_{(0,2)}, \dots\}$ to denote the sequence of tasks that are offloaded to the cloud through the wireless sending channel. For example, if we use the scheduling result in Fig. 3 as the original schedule, we have $S_1 = \{v_4\}$, $S_2 = \{v_6, v_8\}$, $S_3 = \{v_1, v_3, v_5, v_7, v_9, v_{10}\}$, and $S_0 = \{v_2\}$. Suppose that task v_{tar} is executed on the k_{ori}-th core in the original schedule. We know from the outer loop that v_{tar} will be moved onto the k_{tar}-th core in the new schedule. We should derive the new sequence sets S_k^{new} for $0 \leq k \leq K$, which correspond to the sequence of tasks executed (or transmitted) on each core (or the wireless sending channel) in the new schedule. In the linear-time rescheduling algorithm, we will not change the ordering of tasks in the other cores except for the k_{tar}-th core (because we are going to execute task v_{tar} in

this core), i.e.,

$$S_k^{new} = S_k \setminus v_{tar} \text{ for } k = k_{ori}, \tag{18}$$

and

$$S_k^{new} = S_k \text{ for } k \neq k_{tar} \wedge k \neq k_{ori}. \tag{19}$$

In the following, we derive $S_{k_{tar}}^{new}$ by inserting v_{tar} at a "proper" location of the original schedule sequence $S_{k_{tar}}$. We need to satisfy the following task-precedence requirements on the k_{tar}-th core ($k_{tar} = 0$ means the wireless sending channel):

- For any two tasks v_i and v_j that are executed (or transmitted) on the same core or wireless communication channel, task v_i must be executed (or transmitted) before task v_j if task v_i is a transitive predecessor of task v_j in the task graph G.

Hence, we should insert v_{tar} into $S_{k_{tar}}$ such that v_{tar} is executed (or transmitted) after all its transitive predecessors and before all its transitive successors. In order to achieve this goal, we calculate the ready time RT_{tar} of task v_{tar}. RT_{tar} equals to RT_{tar}^{l} (calculated from (4)) when $k_{tar} > 0$ and equals to RT_{tar}^{ws} (calculated from (5)) when $k_{tar} = 0$. In addition, we know the start time ST_i of each task v_i in the original schedule. Therefore, we derive $S_{k_{tar}}^{new}$ as:

$$S_{k_{tar}}^{new} = \{v_{(k_{tar},1)}, \ldots, v_{(k_{tar},m)}, \boldsymbol{v_{tar}}, v_{(k_{tar},m+1)}, \ldots\}, \tag{20}$$

where the start time of tasks $v_{(k_{tar},1)}, \ldots, v_{(k_{tar},m)}$ are earlier than RT_{tar} and the start time of tasks $v_{(k_{tar},m+1)}, \ldots$ are later than RT_{tar}. In this way, it can be proved that the task-precedence requirements on the k_{tar}-th core are preserved.

Now with the new sequence sets S_k^{new} for $0 \leq k \leq K$, we are going to find a new schedule of the task graph in linear time complexity $O(N)$. We maintain two vectors $ready1$ and $ready2$. $ready1_i$ is the number of immediate predecessors of task v_i that have not been scheduled. $ready2_i = 0$ if all the tasks before task v_i in the same sequence S_k^{new} have already been scheduled. In addition, we maintain a LIFO stack for storing the tasks that are ready for scheduling. The stack is initialized by pushing the task v_i's with both $ready1_i = 0$ and $ready2_i = 0$ into the empty stack. We repeat the following steps until the stack becomes empty again.

- Pop a task v_i from the stack.
- Suppose that task $v_i \in S_k^{new}$. If $k = 0$, we schedule the task on the wireless sending channel, and calculate the time when the mobile device completely receives the output data (results) of task v_i from the cloud. Otherwise, schedule the task on the k-th core.
- Update vectors $ready1$ (reducing $ready1_j$ by one for all $v_j \in \mathbf{succ}(v_i)$) and $ready2$, and push all the new tasks v_j with both $ready1_j = 0$ and $ready2_j = 0$ into the stack.

Then we have scheduled all the tasks.

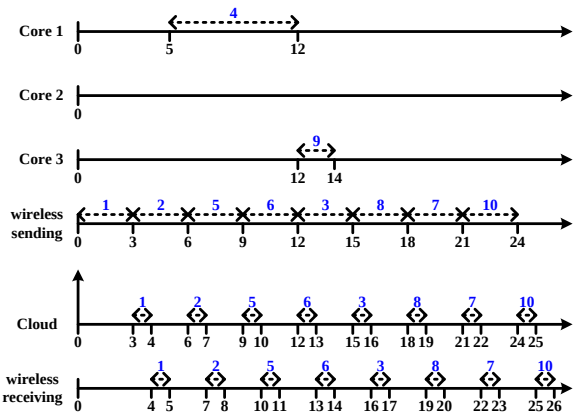


Fig. 4. Task scheduling result generated by the task migration algorithm.

4.2.3 Example of Task Scheduling Result

Fig. 4 presents the task scheduling result generated by the task migration algorithm for the task graph in Fig. 1. The application completion time constraint is set as $T^{total} \leq 27$. The power consumption of cores 1∼3 are set as $P_1 = 1$, $P_2 = 2$, and $P_3 = 4$ under the maximum operating frequency for each core. And the power consumption of the RF components is set as $P^s = 0.5$. Please note that Fig. 3 presents the result of the first step of the MCC task scheduling algorithm (i.e., the initial scheduling algorithm), whereas Fig. 4 presents the result of the second step of the MCC task scheduling algorithm (i.e., the task migration algorithm). Comparing Fig. 3 with Fig. 4, more tasks are offloaded onto the cloud in Fig. 4 for reducing the energy consumption. The application completion time in Fig. 4 is 26 (quite near to $T^{max} = 27$), which is larger than that in Fig. 3. There are still two tasks executed on local cores. The MCC task scheduling algorithm does not offload all the tasks to the cloud due to the application completion time constraint. As can be observed, the wireless sending channel is almost fully occupied. If more tasks are offloaded, the application completion time constraint will be violated. In reality, some tasks can only be executed locally in the mobile device, the MCC task scheduling algorithm can be modified to accommodate these cases easily. In summary, we have $E^{total} = 100.5$ and $T^{total} = 18$ in Fig. 3, and we have $E^{total} = 27$ and $T^{total} = 26$ in Fig. 4. The energy consumption of the application is largely reduced from the result in Fig. 3 to that in Fig. 4. It demonstrates that the task migration algorithm (i.e., the second step of the MCC task scheduling algorithm) can significantly reduce the energy consumption while satisfying the application completion time constraint.

4.3 Step Three: DVFS Algorithm

The initial scheduling algorithm generates a schedule with the minimum application completion time, and the task migration algorithm reduces the energy consumption for executing an application by migrating tasks among the local cores and the cloud. In these two steps of the MCC task scheduling algorithm, we assume the maximum operating frequency for each local core. On the other hand, the DVFS technique enables further energy reduction in the mobile device under the application completion time constraint.

If the DVFS technique is applied to lower the execution frequency of a high performance core such that the high performance core has the same performance as a low performance core, the high performance core still consumes more energy than the low performance core [22]. Therefore, we apply the DVFS algorithm after the task migration algorithm so that the execution unit for each task is determined before the DVFS algorithm.

In the task migration algorithm, the application completion time is sacrificed for reduced energy consumption. In the schedule generated by the task migration algorithm, the actual application completion time is already very close to the deadline T^{max}. There seems no much room left to lower the execution frequency for further reducing energy consumption, as can be observed from the example in Section 4.2.3. However, due to the parallelism requirements on the local cores and the wireless sending channel, i.e., tasks can only be executed or transmitted one by one, the actual start time ST_i of task v_i may be later than its ready time RT_i. Then, the predecessors of task v_i may be executed with lower frequency to make use of the slack between ST_i and RT_i.

A straightforward implementation of the DVFS algorithm is as follows: based on the schedule generated from step two, for each local task we try every execution frequency level in the ascending order and reschedule the tasks until we find an execution frequency that can satisfy the application completion time constraint. If the execution frequency of a task is changed, the whole schedule might be changed due to the task-precedence requirements. We need to reschedule all the tasks after one single change of the execution frequency of a task. Therefore, we propose another implementation of the DVFS algorithm to reduce the time complexity.

Based on the schedule result generated by step two, the proposed DVFS algorithm does not change the schedule of tasks offloaded onto the cloud and the start time of local tasks, in order to avoid frequent rescheduling of all the tasks. The time slack between two consecutive tasks executed on a local core is utilized. The general idea is as following: for each local task, if there is a time slack between the finish time of this task v_i and the start time of the next task v_j on the same local core, we try every execution frequency level in the ascending order (there are a total number M of frequency levels) until we find an execution frequency that (i) does not postpone the start time of task v_j and (ii) does not postpone the start time of the successors of task v_i. Please refer to Algorithm 1 for details.

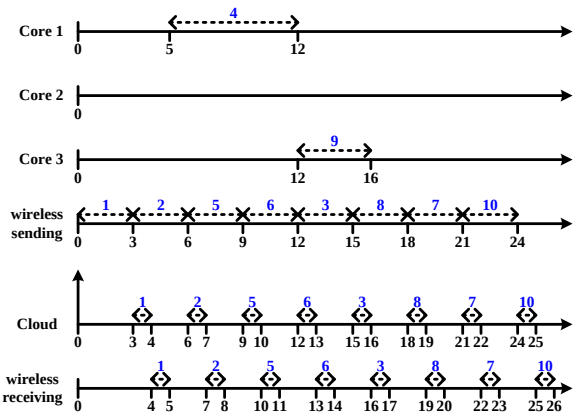


Fig. 5. Task scheduling result generated by the DVFS algorithm.

Algorithm 1 DVFS algorithm

Input: The scheduling result generated by the task migration algorithm.

Output: A task schedule with new execution frequency assignment for local tasks.

1: **for** each local task v_i **do**
2: $flag = 0$; $m = 1$;
3: **while** $flag == 0$ **and** $m < M$ **do**
4: calculate the new finish time FT_i^{new} if v_i is executed using the m-th frequency;
5: **if** $\exists$ next task v_j on the same core **then**
6: $lim_1 = ST_j$;
7: **else** {% task v_i is the last task on this core}
8: $lim_1 = T^{max}$;
9: **end if**
10: **if** $v_i \notin exit\ tasks$ **then**
11: $lim_2 = \min_{v_j \in \mathbf{succ}(v_i)} ST_j$;
12: **else**
13: $lim_2 = T^{max}$;
14: **end if**
15: **if** $FT_i^{new} \leq lim_1$ **and** $FT_i^{new} \leq lim_2$ **then**
16: $flag = 1$;
17: assign the m-th frequency to task v_i;
18: update the finish time of v_i;
19: **end if**
20: **end while**
21: **end for**

We continue using the task graph in Fig. 1 as an example and the task scheduling result generated by the DVFS algorithm is shown in Fig. 5. We assume

there are $M = 3$ operating frequency levels for each core. We set the frequency scaling factors as $a_{k,1} = 0.2$, $a_{k,2} = 0.5$, and $a_{k,3} = 1$ for $k = 1, 2, 3$. And we set $\gamma_k = 2$ for $k = 1, 2, 3$. Comparing Fig. 4 with Fig. 5, the execution frequency of task v_9 is changed from f_k^{max} in Fig. 4 to $0.5 \cdot f_k^{max}$ (k = 3) in Fig. 5, whereas the schedule for the other tasks has not been changed by the DVFS algorithm. We have $E^{total} = 27$ and $T^{total} = 26$ in Fig. 4, and $E^{total} = 23$ and $T^{total} = 26$ in Fig. 5, demonstrating that the DVFS algorithm (i.e., the third step of the MCC task scheduling algorithm) can further reduce the energy consumption while satisfying the application completion time constraint.

4.4 Complexity Analysis of the MCC Task Scheduling Algorithm

Let us analyze the computation complexity of each step in the MCC task scheduling algorithm. (i) Similar to the HEFT algorithm, the computation complexity of the initial scheduling algorithm is $O(E \times K)$, where E is the number of edges in the task graph G, and K is the number of cores. We consider sparse task graphs (i.e., $E = O(N)$ where N is the number of tasks), and therefore the complexity of initial scheduling becomes $O(N \times K)$. (ii) The computation complexity of the task migration algorithm is $O(N^3 \times K)$. (iii) The computation complexity of the DVFS algorithm is $O(N \times M)$, where M is the number of operating frequency levels in each local core. Therefore, the overall computation complexity of the MCC task scheduling algorithm is $O(N^3 \times K)$, which is comparable to the reference work on task scheduling.

5 Experimental Results

We demonstrate the effectiveness of the proposed MCC task scheduling algorithm on a set of generated task graphs with different specifications. We compare the scheduling results of the proposed algorithm to those of the baseline algorithms. All the algorithms are implemented in MATLAB programs executed in a 3.40 GHz Intel Core i7 processor.

We implement four baseline algorithms. Baseline 1 algorithm comprises of only the first two steps of the proposed MCC task scheduling algorithm (without DVFS). Baseline 1 is used to demonstrate the effectiveness of DVFS in energy reduction.

Baseline 2 and 3 algorithms are designed based on exhaustive search. Baseline 2 is as follows:

1) For each task in the task graph, randomly select an execution unit (i.e., on a local core or to the cloud) for it.
2) Generate a schedule for the task graph using an algorithm similar to the initial scheduling algorithm except that the execution unit and frequency for each task have been pre-fixed. (The maximum execution frequency is used for each task.)

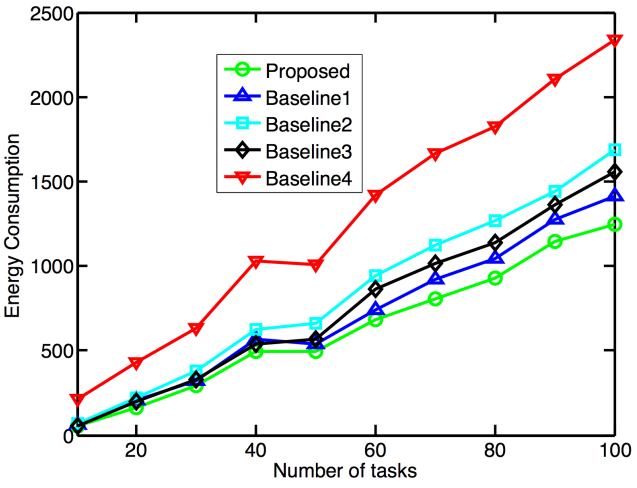


Fig. 6. Energy consumption vs. number of tasks (K=3).

3) Repeat previous steps for 10,000 times to find the schedule with the minimum E^{total} and $T^{total} < T^{max}$.

Baseline 3 has all the steps in baseline 2 and employs the DVFS algorithm (i.e., Section 4.3) after that. Baselines 2 and 3 are designed for demonstrating the effectiveness and efficiency of the proposed algorithm.

Baseline 4 algorithm is similar to the proposed algorithm except that it runs in the *local* mobile device environment only (i.e., the mobile device does not have access to the cloud and only the local resources can be used for task executions.) Baseline 4 is used to prove the benefits of the MCC framework in energy saving and performance enhancement for mobile devices.

A task graph generator is implemented to generate task graphs with various characteristics. The parameters of a task graph are given below.

- Number of tasks in the graph i.e., N.
- Density of edges in the graph i.e., α.
- Number of cores in the mobile device i.e., K.
- Average task computation time on local cores under the maximum operating frequency i.e., T_l^{avg}.
- Average task sending time i.e., T_s^{avg}.
- Average task receiving time i.e., T_r^{avg}.
- Average task computation time on the cloud i.e., T_c^{avg}.

Now we assume there are $K = 3$ heterogeneous cores in the mobile device. Core 1 is a low-power core and core 3 is a high-performance core. The power consumption P_k values of the three cores under the maximum operating frequency are set as $P_1 = 1$, $P_2 = 2$, and $P_3 = 4$. The power consumption of the RF components is set as $P^s = 0.5$. We assume there are $M = 4$ operating frequency levels for each core. The frequency scaling factors are $a_{k,1} = 0.2$, $a_{k,2} = 0.5$, $a_{k,3} = 0.8$, $a_{k,4} = 1.0$ for $k = 1, 2, 3$. We set $\gamma_k = 2$ for $k = 1, 2, 3$.

Ten task graphs with different task numbers N and

different characteristics are generated for comparing the proposed algorithm with baseline algorithms. We plot in Fig. 6 the energy consumption of the mobile device for executing an application as a function of the number of tasks in the application. We can observe that (i) comparing with baseline 1, the proposed algorithm achieves energy reduction due to the effect of the DVFS algorithm (the energy reduction due to DVFS is in the range of 7.5%∼19.6%), (ii) comparing with exhaustive search-based baseline 2 and 3, the proposed algorithm also achieves energy reduction (the maximum energy reduction are 28.1% and 20.7%, respectively, compared with baseline 2 and 3), and (iii) comparing with baseline 4, the proposed algorithm achieves huge energy reduction (in the range of 45.7%∼74.9%) due to the MCC framework.

Table 1 shows the application completion time T^{total} of the scheduling results from all the algorithms. Except baseline 4, all the other algorithms can generate scheduling results satifying the application completion time constraint. Table 1 also compares the program execution time of the proposed algorithm and baseline 1, 2, 3. The execution time of the proposed algorithm is a little bit longer than baseline 1, demonstrating the small computation overhead of the DVFS algorithm. In addition, the proposed algorithm significantly reduces the execution time compared with the exhaustive search-based baseline 2 and 3.

Also, we use a realistic task graph from [23] i.e., the face recognition task graph and set the application completion time constraint as $T^{total} \leq 70$. Baseline 1 achieves $E^{total} = 95$ and $T^{total} = 69$, and the proposed algorithm achieves the same T^{total} but a lower $E^{total} = 75.9$ due to the DVFS algorithm. Baseline 2 achieves $E^{total} = 131$ and $T^{total} = 65$, baseline 3 achieves $E^{total} = 112.2$ and $T^{total} = 69$, and baseline 4 achieves $E^{total} = 445$ and $T^{total} = 95$. From the above results we can observe that the proposed algorithm can achieves the lowest E^{total} with the application completion time constraint satisfied.

Furthermore, we assume there are $K = 6$ heterogeneous cores in the mobile device. Core 1 is a low-power core and core 6 is a high-performance core. The power consumption P_k values of these cores under the maximum operating frequency are set as $P_1 = 1$, $P_2 = 2$, $P_3 = 4$, $P_4 = 8$, $P_5 = 16$, $P_6 = 32$. The other parameters are same as before. Fig. 7 shows the energy consumption of the mobile device using different algorithms. We can observe similar results as in the previous case: the proposed algorithm always achieves the lowest energy consumption compared with baseline algorithms. Please note that the benefit of the MCC framework in energy saving is not as prominent as before (comparing the proposed algorithm with baseline 4), because the mobile device has stronger computing resources in this case.

Table 2 shows the application completion time of the scheduling results from all the algorithms and also

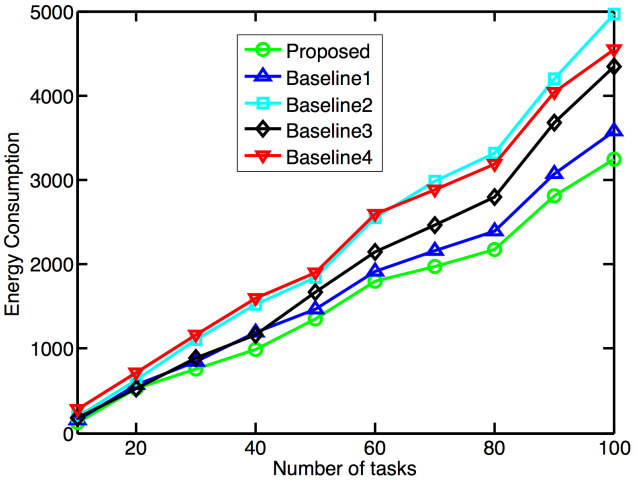


Fig. 7. Energy consumption vs. number of tasks (K=6).

the program execution time of the proposed algorithm and baseline 1, 2, 3. Again due to the stronger computing resources in the mobile device, even baseline 4 can fulfill the application completion time constraint. In addition, the proposed algorithm demonstrates superior efficiency compared with exhaustive search-based algorithms.

6 Conclusion

This work studies the MCC task scheduling problem. To our best knowledge, this is the first task scheduling work that minimizes energy consumption under hard completion time constraints for the applications in the MCC environment, taking into account the joint task scheduling on the local cores in the mobile device, the wireless communication channels, and the cloud. A novel algorithm is proposed that starts from a minimal-delay scheduling solution and subsequently performs energy reduction by migrating tasks among the local cores and the cloud and by applying the DVFS technique. A linear-time rescheduling algorithm is proposed for the task migration such that the overall computation complexity is effectively reduced. Simulation results demonstrate significant energy reduction with the application completion time constraint satisfied.

Acknowledgments

This work is supported in part by a grant from the Software and Hardware Foundations of the National Science Foundation.

References

[1] X. Lin, Y. Wang, Q. Xie, and M. Pedram "Energy and performance-aware task scheduling in a mobile cloud computing environment," Proc. IEEE International Conf. Cloud Computing (Cloud '14), pp. 192-199, Jun. 2014, doi: 10.1109/CLOUD.2014.35.

TABLE 1
Comparison Between the Proposed Algorithm and the Baseline Algorithms ($K = 3$)

N	T^{max}	T^{total}					Exe. Time (s)			
		Prop.	Base1	Base2	Base3	Base4	Prop.	Base1	Base2	Base3
10	100	98	98	98	98	97.25	0.02560	0.02510	2.19173	2.19213
20	120	120	120	118	118	119	0.04224	0.04169	4.19182	4.19214
30	150	150	150	148	148	148	0.10432	0.10386	6.38857	6.38887
40	180	180	180	176	178.25	180	0.13905	0.13858	8.75305	8.75339
50	200	199	199	200	200	200	0.31020	0.30971	11.12985	11.13021
60	230	230	230	225	229	229	0.45234	0.45188	14.02819	14.02851
70	250	250	250	249	249	249	0.43461	0.43424	16.31539	16.31572
80	280	280	278	278	278	279.25	0.86852	0.85867	19.53377	19.53411
90	300	300	300	295	295	306	0.86966	0.85923	22.73272	22.73305
100	320	319.25	318	317	318.25	337	1.07801	1.06775	26.03246	26.03278

TABLE 2
Comparison Between the Proposed Algorithm and the Baseline Algorithms ($K = 6$)

N	T^{max}	T^{total}					Exe. Time (s)			
		Prop.	Base1	Base2	Base3	Base4	Prop.	Base1	Base2	Base3
10	60	60	60	54	55.5	59	0.06748	0.06693	2.08629	2.08644
20	70	70	70	66	67.25	70	0.32003	0.30977	3.99896	3.99930
30	90	89	89	84	88.75	90	0.64587	0.63581	6.28821	6.28853
40	100	100	100	99	99	100	0.91461	0.90420	8.46932	8.46944
50	120	119	119	120	120	119	1.74727	1.73733	10.75493	10.75507
60	130	129.25	128	130	130	130	1.93840	1.92801	14.07699	14.07732
70	160	160	160	155	160	160	2.91528	2.90505	16.36597	16.36627
80	170	170	170	164	168	170	3.52556	3.51517	18.76348	18.76385
90	180	180	180	177	179	180	5.06953	5.05901	22.41015	22.41033
100	200	200	200	200	200	200	6.87788	6.86762	24.87639	24.87698

[2] S. Chen, Y. Wang, and M. Pedram "A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud" Proc. IEEE Global Communications Conference (GLOBECOM '13), pp. 2885-2890, Dec. 2013, doi: 10.1109/GLOCOM.2013.6831512.

[3] B. Hayes, "Cloud computing," Communications of the ACM, vol. 51, no. 7, pp. 9-11, Jul. 2008, doi: 10.1145/1364782.1364786.

[4] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities," Proc. IEEE International Conf. High Performance Computing and Communications (HPCC '08), pp. 5-13, Sep. 2008, doi: 10.1109/HPCC.2008.172.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50-58, Apr. 2010, doi: 10.1145/1721654.1721672.

[6] H. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," Wireless Communications and Mobile Computing, vol. 13, no. 8, pp. 1587-1611, Dec. 2013, doi: 10.1002/wcm.1203.

[7] A. Khan and K. Ahirwar, "Mobile cloud computing as a future of mobile multimedia database," International Jour. Computer Science and Communication, vol. 2, no. 1, pp. 219-221, 2011.

[8] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," Proc. the 2nd USENIX Conf. Hot Topics in Cloud Computing (HotCloud '10), pp. 4-4, Jun. 2010.

[9] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: a partition scheme," Proc. International Conf. Compliers, Architecture, and Synthesis for Embedded Systems (CASES '01), pp. 238-246, 2001, doi: 10.1145/502217.502257.

[10] K. Kumar, J. Liu, Y. H. Lu, and B. Bhargave, "A survey of computation offloading for mobile systems," Mobile Networks and Applications, vol. 18, no. 1, pp. 129-140, Feb. 2013, doi: 10.1007/s11036-012-0368-0.

[11] U. Kremer, J. Hicks, and J. Rehg, "A compilation framework for power and energy management on mobile computers," *Languages and Compilers for Parallel Computing*, Springer-Verlag Berlin Heigelberg, vol. 2624, pp. 115-131, 2003.

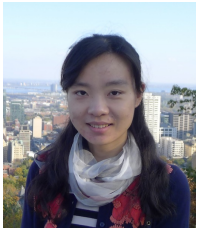
[12] A. Cheung, S. Madden, O. Arden, and A. C. Myers, "Automatic partition of database applications," Journal Proc. the VLDB Endowment, vol. 5, no. 11, pp. 1471-1482, Jul. 2012, doi:10.14778/2350229.2350262.

[13] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, Aug. 2002, doi: 10.1109/71.993206.

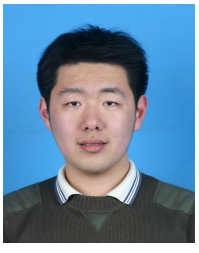
[14] S. C. Kim, S. Lee, and J. Hahm, "Push-pull: determinisitic search-based DAG scheduling for heterogeneous cluster systems," IEEE Trans. Parallel and Distributed Systems, vol. 18, no. 11, pp. 1489-1502, Nov. 2007, doi: 10.1109/TPDS.2007.1106.

[15] M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," Proc. the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11), pp. 43-56, Jun. 2011, doi: 10.1145/1999995.2000000.

[16] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," ACM SIGMETRICS Performance Evaluation Review, vol. 40, no. 4, pp. 23-32, Mar. 2013, doi: 10.1145/2479942.2479946.
[17] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," Proc. Asia and South Pacific Conf. Design Automation (ASP-DAC '06), pp. 473-478, Jan. 2006, doi: 10.1109/ASP-DAC.2006.1594730.
[18] Z. Li, C. Wang, and R. Xu, "Task allocation for distributed multimedia processing on wirelessly networked handheld devices," Proc. International Parallel and Distributed Processing Symposium (IPDPS '02), pp. 1-6, Apr. 2002, doi: 10.1109/IPDPS.2002.1015589.
[19] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," Proc. the 9th IEEE/ACM International Symp. Cluster Computing and the Grid (CCGRID '09), pp. 92-99, May 2009, doi:10.1109/CCGRID.2009.16.
[20] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: can offloading computation save energy?" Computer, vol. 43, no. 4, pp. 51-56, Apr. 2010, doi: 10.1109/MC.2010.98.
[21] Nvidia Corporation. http://www.nvidia.com/object/tegra-4-processor.html
[22] P. Greenhalgh, "Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, 2011.
[23] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing" Proc. International Conference on Cloud Networking (CLOUDNET '12), pp. 80-86, Nov. 2012, doi: 10.1109/CloudNet.2012.6483660.



Xue Lin (S'12) received her B.S. degree from Tsinghua University, Beijing, China, in 2009. She is now a Ph.D. student in the Department of Electrical Engineering at University of Southern California. Her advisor is Prof. Massoud Pedram. She has been working on power management of photovoltaic (PV) systems, near-threshold computing of FinFET circuits, task scheduling in real-time and mobile systems, and power management of energy storage devices and hybrid electric vehicles. She has published more than 40 papers in these areas. She received the best paper award at 2014 IEEE International Symposium on VLSI (ISVLSI).



Yanzhi Wang (S'12) received his B.S. degree with distinction in electronic engineering from Tsinghua University, Beijing, China, in 2009, and Ph.D. degree in electrical engineering at University of Southern California, in 2014, under the supervision of Prof. Massoud Pedram. He is currently a postdoctoral research associate and (part-time) lecturer at University of Southern California. His current research interests include system-level power management, next-generation energy sources, hybrid electrical energy storage systems, near-threshold computing, digital circuits power minimization and timing analysis, cloud computing, mobile devices and smartphones, electric vehicles and hybrid electric vehicles, and the smart grid. He has published around 130 papers in these areas. He received the best paper awards at 2014 IEEE International Symposium on VLSI (ISVLSI) and 2014 IEEE/ACM International Symposium on Lower Power Electronics Design (ISLPED).

Qing Xie (S'12) has received his PhD degree in Electrical Engineering Department at University of Southern California, under the supervision of Prof. Massoud Pedram. His research interests are in the area of thermal modeling and management, system-level power management, low-power design and management of energy storage systems and mobile devices, and near-threshold VLSI circuits design. He has received the Best Paper Award from the 30th IEEE International Conference on Computer Design.

Massoud Pedram (F'01), who is the Stephen and Etta Varra Professor in the Ming Hsieh department of Electrical Engineering at University of Southern California, received a Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 1991. He holds 10 U.S. patents and has published four books, 13 book chapters, and more than 140 archival and 380 conference papers. His research ranges from low power electronics, energy-efficient processing, and cloud computing to photovoltaic cell power generation, energy storage, and power conversion, and from RT-level optimization of VLSI circuits to synthesis and physical design of quantum circuits. For this research, he and his students have received seven conference and two IEEE Transactions Best Paper Awards. Dr. Pedram is a recipient of the 1996 Presidential Early Career Award for Scientists and Engineers, a Fellow of the IEEE, an ACM Distinguished Scientist, and currently serves as the Editor-in-Chiefs of the ACM Transactions on Design Automation of Electronic Systems and the IEEE Journal on Emerging and Selected Topics in Circuits and Systems. He has also served on the technical program committee of a number of premiere conferences in his field and was the founding Technical Program Co-chair of the 1996 International Symposium on Low Power Electronics and Design and the Technical Program Chair of the 2002 International Symposium on Physical Design.