Energy-Aware Real-time Scheduling on Heterogeneous Multi-Processor

Gang Wang, Wenming Li, Xiali Hei
CSE Department, University of Connecticut, Storrs, CT 06269
Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190
CSIT Department, Frostburg State University, Frostburg, MD 21532
g.wang.china86@gmail.com wanggang@ict.ac.cn

Abstract—**Heterogeneous Multi-Processor has been steadily gaining importance because of its potential energy efficiency compared to homogeneous multi-processor architectures. Many challenges for scheduling algorithms on heterogeneous multi-process are created during the shift from homogeneous multi-processor to heterogeneous architectures, especially for real-time applications . In this paper, we introduce an optimal algorithm, using energy-aware real-time scheduling, for multiple tasks on heterogeneous multi-core processors. We first propose an energy modelling from both dynamic and static energy perspectives, considering the parameters both of software and hardware levels. After that we propose a real-time scheduling policy of Optimal Job to a Fast Processor First (OJFPF) using the parameter of energy achieved from energy modelling, the length of tasks as well as the local priority to assign the priority to each task. Our simulation results show that OJFPF algorithm using the energy modelling significantly reduce the overall energy consumption and improve the performance.**

I. Introduction

Due to the increase in transistor budgets enabled by Moore's law, multi-core processing architectures are now being increasingly used in embedded systems with the real-time constraints, as the result the methods for real-time scheduling have been gaining significant importance. Conventional multi-core processor consists of the identical cores, which the cores in processor are symmetric. Besides the symmetric multi-core processors, homogeneous and heterogeneous multi-core processors also gain in popularity, whose motivations are driven by using cores geared to perform specific tasks well and cheap. An alternative design approach for multi-core processors, aimed for low-power and high performance, is to implement heterogeneous cores on a processor which provide a promising solution for power-efficient computing [1] [2][3].

The real-time community has recognized the trends of substantially using heterogeneous multi-core processors and provides solutions for this trends. However, when modelling power and energy models, it makes, in most cases, the simplifying assumptions, limiting the applicability of the presented solutions. There exists too many common assumptions in literature, such as the energy consumption of different applications only related to a function of execution time rather than the task characteristics, the scheduling algorithms only considering the time-stamp of tasks. To provide a better scheduling for the tasks, it should not only consider the software-level parameters, such as execution time, but also utilize the other parameters of executing tasks, such as the characteristics of memory accessing as well as the requirements of deadline and critical regions.

To schedule the multi-task assignments in heterogeneous multi-core processors, there exists a larger number of algorithms proposed to deal with the schedulability of these multi-task assignments. Priority-based real-time multi-processor scheduling approaches are state-of-the-art broadly used. A scheduler of real-time system assigns a fixed priority to all the tasks and then schedules them to execute on one or more processor(s). In the uniprocessor, several optimal scheduling algorithms exist to schedule the tasks in simply ways, such as Deadline-Monotonic(DM) algorithm [4], Earliest Deadline First(EDF) scheduling [5] as well as Least Laxity First(LLF) scheduling [6]. However, these scheduling methods are not the optimal algorithms for executing the multi-task assignments on heterogeneous multi-core processors.

Traditional algorithms for scheduling the tasks aim to reduce the power consumption of the system by assigning the tasks to their favourite processors, usually ignoring other factors, such as the deadlines of tasks and the uses of critical regions. The managements of the tasks scheduling depend on the properties of the tasks as well as the characteristics of processors. Muhammad and Stefan [7] proposed an energy-aware partitioning method to map the tasks onto a heterogeneous multi-core platform using the Least Loss Energy Density Algorithm(LLED) and the sleep states. Although Muhammad and Stefan's method achieved the effectiveness of reducing energy, however, their method is complex, having a large computations to assign the tasks, as well as only from the energy-aware aspect to consider the scheduling. Jason and Bo [1] proposed a scheduling approach that maps the programs to the most appropriate core, based on the program phases. Their approach achieves a good experimental result only in the power model of energy delay product, without considering other factors. Robert and Marko [8] presented a optimal fixed priority scheduling with deferred pre-emption to determine both the priority orderings of tasks and the lengths of their final non-pre-emption regions. Their algorithm is optimal for fixed priority scheduling and assumes that there exists a schedulable combination. Nagesh *et.al* [9] evaluated two scheduling algorithms, longest job to a fast processor first and critical job to a fast processor first separately. However, the authors only consider one parameter for each scheduling algorithm to evaluate their experimental results.

Above mentioned scheduling approaches and algorithms mainly consider one or limited parameter(s) to design a real-time scheduling on multi-core processors taking energy-aware into account. However, it should consider and utilize several significant parameters together to design an optimal scheduling algorithm. In this paper, we introduce an optimal algorithm for energy-aware real-time scheduling on heterogeneous multi-core processors. We first propose a energy modelling from both dynamic and static energy perspectives, considering the parameters both from software and hardware levels. After that we propose a real-time scheduling policy of Optimal Job to a Fast Processor First (OJFPF) using parameters of energy achieved from energy modelling, the length of tasks as well as local priority to assign the priority to each task.

The remainder of this paper is organised as follows. Section II proposes the energy model to evaluate the energy consumption. Section III describes the real-time scheduling algorithm of OJFPF. Section IV shows our evaluation methodology and experimental results. Section V concludes this paper.

II. Energy Modelling

This paper studies the priority-based scheduling of a set of sporadic tasks (or *taskset*) on the heterogeneous multi-processor. The heterogeneous processors achieve significant improvements both in performance and energy consumptions. Here we first propose a model to evaluate the energy consumptions, then followed by the scheduling policy of *optimal job to a fast processor first(OJFPF)*.

Modelling energy for the real-time scheduling is a challenging task. It needs to consider the processes designed over the time span of more than a decade. To model energy, it should exploit the variability in demand as well as the characteristics of the input computational capacities, both within a given thread and across threads. The variations in computational activity are often repetitive because of loop-oriented execution semantics [10]. Energy depends not only on the computational activities, but also on the scheduling policy used on the single or multiple processor(s). Here we propose an energy modelling framework to account for these factors together.

The energy model used in state-of-the-art usually considers two different parts: dynamic energy and static energy. Assuming, indeed, all processors execute the same instruction set, however, utilize different resources and achieve different performance and energy efficiency on the same task. One of the major tasks of operating system is to map the tasks to different processors, attempting to meet the pre-defined objective. The processors in the heterogeneous multi-processor should provide a wide and spaced range of the complexity/performance design space to cover the requirements of application execution characteristic and system priority instruction used. With these requirements, it should consider how to choose the processors to minimize the energy consumptions. Due to typical programs with the different execution characteristics of instructions, the best processor during one phase may not be the best one for the next phase. It needs to full considerations to minimize the switches among processors in the overall perspectives.

Dynamic energy consumption usually relates to the frequency and instruction sets, and static energy consumption is often considered as a constant factor related with the physical aspects of the processors. For the overall energy consumption, it assumes that execution time of an application is the only factor to determine the energy consumption. However, even two tasks with the identical execution time, it may consume significantly different energy due to both the characteristics of instructions, different instructions executing on different parts of CPU, and the number of cache misses involved. In real-time scheduling level, the static energy consumption is not only related with the physical aspects, but also highly bounded up with the time slices of the sleep states. It should further define a more refined and accurate energy consumption model to take full advantages of the sleep states' characteristics of the processor.

This paper assumes that the frequency, or speed, per processor is firmed, which takes advantages of the computer architectures of Globally Asynchronous Locally Synchronous (GALS) [11] architectures. Similarly, this paper also assumes that each processor has the firm, maybe different for different processors, sleep states which are the states of low power consumption.

From the view of the hardware performance, several key hardware performance parameters are related to the energy consumption closely. We identify five hardware-type parameters as the key parameters to evaluate the energy consumption, which are *Performance, Instructions, Cache, TLB, Branches*. The parameter of *Performance* mainly relates to the Cycles of application, and *Instructions, Branches* relate to the type and number of different instructions of application, however, *Cache, TLB* are mainly bounded up with the memory issues, such as the number of memory accesses and memory misses. Usually, memory issues take an important part in the these parameters. We treat the execution time of applications and the number of sleep states of each processor as the software performance parameters in this paper. Table I shows the parameters that affect the energy modelling.

TABLE I
Parameters related with Energy Modelling

Hardware	***Software***
Performance Instructions	Excution Time
Cache TLB Branches	Sleep States

Fig. 1 shows the conceptual structures of a hierarchical energy consumption modelling to generate energy orders. The energy orders will be used as a factor to design the scheduling algorithm later.

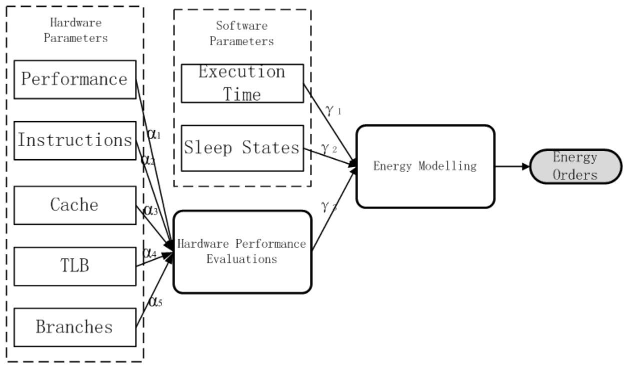

Fig. 1. Energy Modelling

The preference of the task to a certain processor should not only be set with respect to the energy consumption, meaning that the most favourite processor for a task is the one where its energy consumption is minimal, otherwise, it may cause the unpredicted and/or unintended results. It should consider other issues, such as the deadlines of the task and the time duration in the critical area.

III. Optimal Job to a Fast Processor First (OJFPF)

In the previous section, we illustrated that the energy consumption plays an important role to assign the Jobs/Tasks to processors, however, the energy modelling is not the only factor to determine a good operating system scheduler for the real-time scheduling algorithm. It should consist of other factors for a good one. We propose a real-time scheduling, called *Optimal Job to a Fast Processor First (OJFPF)*, which also bases on the assigned priority to schedule the tasks on processors. In this section, we pose other three key parameters relating to the assignments of priorities under OJFPF. We then derive a tractable algorithm using these parameters . To make it more sensible, the terms of *job* and *task* have the same meanings in this paper, both which can be either the threads or processes.

A. Length of Tasks

The length of a task, commonly called execution time, is one factor of OJFPF. A task of application is the work assigned to each thread, which can execute on different processors to finish a certain function. Also, the application could provide the relative information of task length. The length of a task is determined by the number of iterations in this paper, such as loop functions. We estimate the length of a task based on the number of iterations assigned to each thread, because it is impossible to predict the exact execution time at compile time. However, using the system calls, it can send the relative task length information to the operating system kernel by revising the application before a thread is created. By statical analysis, the application can easily obtain the number of iterations for each thread at the compile time.

In the sense of efficiencies of the overall multi-processor, the longest job/task, only from the perspective of task length, should be assigned to the fast processor to improve the overall performance. The length of a job can be estimated by the iterations just mentioned above.

B. Local Priority Parameters

Multi-thread systems concern with the resources sharing, which are different from the single threaded systems, because of the interactions and communications among the threads. Usually, these shared resources can only be used by one task at a time, and the use of shared resource cannot be interrupted if one task is using it. Such resources are said to be serially reusable, which may include certain peripherals, shared memory, and the CPU. The CPU protects itself against the simultaneous uses through synchronization mechanism, such as semaphores and locks. There exists a code section, which called a critical region, to keep the synchronization use. If two or more threads enter into the same critical region simultaneously, a catastrophic error can happen.

One of the most common operations for the critical region is waiting during the thread interactions, such as waiting for entering(acquiring a lock/semaphores operation) or waiting for all the threads to finish(barrier operation). This waiting would definitely waste a lot of CPU time, especially for the thread taking a long time to use the critical region and other remaining threads doing nothing except for the waiting. For those threads that cannot acquire a lock to enter critical region, the operating systems have no choice but to put them into the sleep state. When the thread that had the lock releases the excludable lock, it wakes up a waiting thread. Also, this process needs to reload certain necessary parameters to enter the critical region. When a certain thread wakes up, the scheduler of operating system sends the thread to an idle processor to execute.

From the above descriptions of critical region, it would be better to send a thread with the longer critical region to a fast processor from the view of waiting of thread interactions, so that this scheduling reduces the overall sleep states of threads sets.

Real-time processors are often specified as having a start time(release time) and a stop time(deadline). The release time can be expressed as the time at which the process must start after some events occurred that triggered the process. Also, the deadline is the time by which the task must complete, so the scheduler must allot enough CPU time to the process so that the related task can indeed complete. There usually exists two deadlines: hard deadline and soft deadline. A hard deadline is the one in which there is no value to the computation if it completes after the deadline, while a soft deadline is the value of a late result diminishes but does not immediately disappear with time.

C. Local Priority Assignment

Based on the needs of critical regions and the deadline of tasks, we first assign local priorities to the tasks. This assignment is called local priority assignment, which is based on local data(critical region requirements and deadline of tasks). This priority assignment problem is proven to be a NP-complete problem [12]. Finding an optimal solution is generally not feasible for the large size problems. Here we use a heuristic to solve this problem.

The flow of the heuristic for local priority is shown in Fig. 2. The basic idea is to define the local deadlines of tasks over iteration steps, then assign the local priorities based on the deadlines and utilizations of critical regions. Intuitively, earlier deadlines obtain higher priorities and longer local deadlines assign lower priorities. Meanwhile, the higher utilization of the critical region for tasks, the higher of priorities.

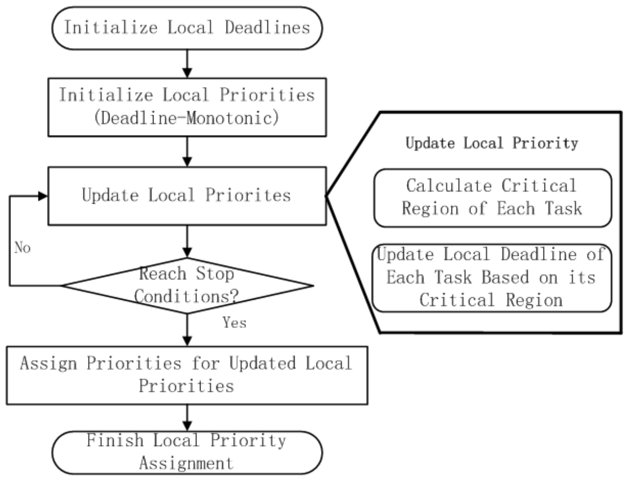
Initialize Local Deadlines
Initialize Local Priorities (Deadline-Monotonic)
Update Local Priorites
No
Reach Stop Conditions?
Yes
Assign Priorities for Updated Local Priorities
Finish Local Priority Assignment
Update Local Priority
Calculate Critical Region of Each Task
Update Local Deadline of Each Task Based on its Critical Region

Fig. 2. Local Priority Assignment Algorithm

Algorithm 1 Update Local Priority(K1)

```
1: Initialize the critical region $\epsilon$ of every task
2: for all task $\tau_i$ do
3:     $\Delta c_{\tau_i} = t_{\tau_i} * (1 - \Sigma_{\tau_i \in T(e)} c_{\tau_i} / t_{\tau_i})$
4:     $c_{\tau_i} = c_{\tau_i} + \Delta c_{\tau_i}$
5: end for
6: for all task $\tau_j \in (lp(\tau_i)) \cup \{\tau_i\}$ do
7:     update $r_{\tau_i} and \quad deadlines$
8: end for
9: for all task $\tau_i$ do
10:     $\epsilon^N_{\tau_i} = \epsilon_{\tau_i} / max_{\tau_i \in T(e)} \epsilon_{\tau_i}$
11:     $d_{\tau_i} = d_{\tau_i} * (1 - K_1 * \epsilon^N_{\tau_i})$
12: end for
```

Initially, the deadlines of tasks are the same as their periods which real-time processors require. After initialization, deadlines of tasks are modified and the priorities are assigned using the approach of Deadline-Monotonic [14] [13]. However, there no exists guarantee that the deadline-monotonic policy is optimal for multi-tasks running on multi-processor with non-preemptive critical regions, and there is no optimal counterpart that can be used here. The deadline-monotonic mechanism together with critical region restriction are a sensible choice in the context of this heuristics.

During the iterations, deadlines and local priorities of tasks are updated simultaneously, as shown in Algorithm 1.

Before talking about Update Local Priority Algorithm, it is necessary to give some definitions to fully understand this algorithm. $T = \{\tau_1, \tau_2, ..., \tau_n\}$ is the set of tasks that perform the computations, where τ_i represents a periodically activated task with period t_{p_i}. ϵ represents the value of critical region. Here, we use the execution time of a task executing on a critical region as ϵ. The critical region of a task reflects how much the response times are affected by extensions in the execution times of other tasks. c_{τ_i} is a worst case computation time of task τ_i; r_{τ_i} represents the corresponding response time and d_{τ_i} represents the corresponding deadline. Δc_{τ_i} is the increasement of execution time after the execution of critical region. $lp(\tau_i)$ means the set of tasks with priorities lower than task τ_i. First, we initialize the critical region ϵ of every task according to their executing time on the critical regions. Then, we recalculate the increase of execution time for each task. After obtaining the c_{τ_i} , the response time and deadlines of task τ_i and of lower priority tasks on the same critical region as τ_i is also recomputed. The numerical values of critical region are normalized, obtaining a value ϵ^N for each task, and finally the local deadlines are computed as $d = d * (1 - K_1 * \epsilon^N)$.

The parameter K_1 is initially set to 1 here, then if needs, adjusted in the later iteration steps using the complex strategies, such as taking the number of iteration steps of entering critical region and the number of times the priority assignment remains unchanged into account. After updating the local deadlines, it checks the stop conditions. The stop conditions can be varied according to the actual requirements of tasks, such as the number of iterations steps for each task. If the stop conditions reached, the priority assignment will finish, otherwise, it keeps iterating. After the update local priority algorithm, we assign the local priority according to their updated deadlines which consider the requirements of critical regions.

D. Regression Modelling

There usually exists hundreds or thousands of tasks waited for executing on multi-core processors. So many tasks compete for the limited processors, to keep them schedule and cooperate each other correctly, these tasks need to assign multi-priority to determine the orders of execution and the processors that are best suitable to execute for them from the overall perspectives.

As we presented above, several factors together affect the scheduling of threads on multi-processor. Regression modelling, usually used in machine learning areas, has been proposed to predict performance and priorities [15]. Here, we use regression modelling to assign the priorities to the waiting scheduled tasks to achieve much better overall performance. We identify the key performance parameters that are most related to the energy consumption of the scheduled tasks. The three selected key performance parameters include *energy* provided by energy modelling in Section II, the length of task which can be expressed as execution time *execution_time* as well as the local priority *local_priority*. After identifying the three key performance parameters, we specify a linear regression model as shown in equation (1) to assign the priority of each task with respect to the evaluated priorities. β_1, β_2, β_3 denote the corresponding regression coefficients, which can be interpreted as the expected change in the priorities for multiple tasks in energy modelling, execution time as well as the local priority.

$$\begin{aligned} priority &= \beta_0 + \beta_1 \times energy \\ &\quad + \beta_2 \times execution\_time \\ &\quad + \beta_3 \times local\_priority \end{aligned} \tag{1}$$

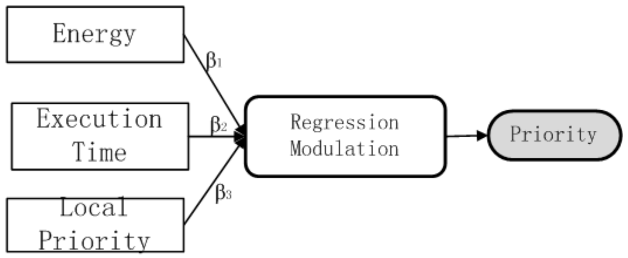

Fig. 3. Regression Model to Assign Priority

Based on the predicted priorities, we propose a new scheduling policy, called OJFPF. The basic idea of OJFPF is that when a task has higher assigned priority than others, the scheduler sends this task to a fast processor. Tasks can usually have different and unique priorities due to the processes of assigning priority by linear regression model. However, if there are two or more tasks have the same priority, here we set energy consumption prior to local priority which is prior to execution time to assign the processor. If all three parameters are the same for the scheduled tasks having highest priority, we randomly select a task to a fast processor.

IV. Evaluations and Results

In order to evaluate the effectiveness of our scheduling algorithm on heterogeneous Multi-Processor, we conduct our experiments on the Loongson-3B1500 Platform, which is designed by Institute of Computing Technology, Chinese Academy of Sciences. This Multi-Processor maintains eight-core architectures, which the operating frequency ranges from 1.0 GHz to 1.5 GHz, depending on the core voltage supplies varied from 1.0 V to 1.3 V [16].

In the experiments, we modify several applications so that they send the task-related and system-related information to the kernel and scheduler using the system calls before a thread is created. Here we give an example of modification. When a task calls *pthread_create()* to create a thread, it calls a serial of system calls *sys_clone()*, *do_fork()* as well as *copy_process()* functions sequentially. To schedule a newly created thread, the system scheduler uses the *sched_fork()* function. After modification, it passes task information to the scheduler using new system calls and modify both *copy_process()* and *sched_fork()* functions to use this information.

To derive the parameters information and regression model in section II and III, we need to develop the training data to train the model. To generate these training data, we randomly select 8 benchmarks from SPEC CPU206 [17], which include 4 integer benchmarks(*bzip2, gcc, mcf and astar*) and 4 floating point benchmarks(*gamess, milc, povray and lbm*). We collect the hardware performance data and task-related data separately. The hardware performance data is collected by hardware performance counters using the Linux Perfctr Library, and the task-related data is collected with the tools of Pin [18] which both are customized program analysis tools with the dynamic instrumentations. With this training data, we evaluate and build the parameters and regression model by using statistical package R [19] which is used for statistic computing.

We compared the performance of our scheduling scheme with other popular scheduling algorithms. Earliest Deadline First(EDF) scheduling is an optimal scheduler for fully preemptive sporadic tasks with constrained deadlines on a single processor [20]. Deadline-Monotonic(DM) algorithm is also excellent real-time scheduling algorithm related with deadline requirements [20]. With the same parameters and regression model, we randomly select other four SPEC CPU2006 benchmarks(*gobmk, omnetpp, cactusADM and GemsFDTD*) to evaluate the our scheduling as well as other scheduling algorithms. We revise the benchmarks interfaces and bind these four randomly selected benchmarks as one test set sequentially. In this experiment, we use the number of iterations (*#iterations*)of the revised test set to represent the workload of tasks, and the number of iterations range from 1 to 100.

To better show the uniform performance of our scheduling, we use the normalized methods to normalize the corresponding results, ranging from 0% to 100%, such as Total Energy, Execution Time as well as the number of tasks shifts among processors. Equation(2) shows the normalized method, where $R(a)$ is the result set for the type of parameter a. Here we treat the maximum metric of each measurement as the base line when the number of iteration is equal to 1.

$$a_N = a/max_{a \in R(a)} a \tag{2}$$

In our experiments, we evaluate the results from three perspectives, which are the normalized total energy consumption, the normalized execution time as well as the number of tasks shifts among the processors. The reason why we select the normalized methods is that we conduct extensive simulations and the range of results data is very wide. If drawing so wide data in one direction, the figure is very vague and hard to understand.

From Fig. 4, it can be seen that when the number of iteration is less than 8, the number of processors of multi-processor, the total energy consumptions are almost the same for EDF, DM as well as our method OJFPF. However, when the number of iterations are larger than 8, we could see that the rates of increases for EDF and DM are very high than OJFPF. As the increase of the number of iterations, the energy consumed by OJFPF algorithm is much more stable and less than that of others.

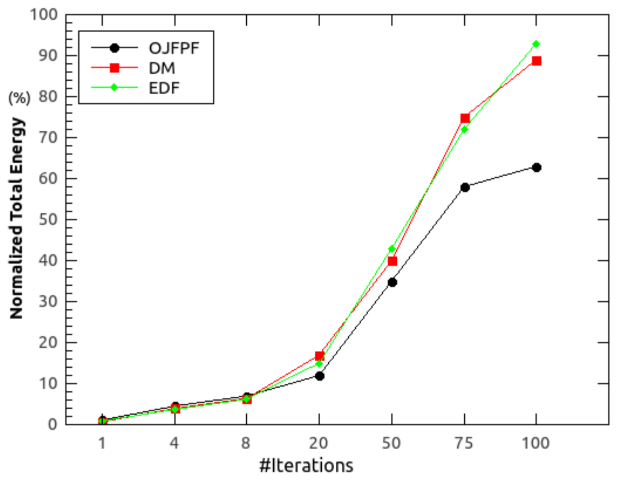

Fig. 4. The Trend of Normalized Total Energy

For the execution time with respects to the number of iterations, shown in Fig. 5, the trend is almost the same with the trend of energy consumption. However, it can be seen that our method is much stable compared with others. This also indicate the energy consumption has the direct relation with the execution time.

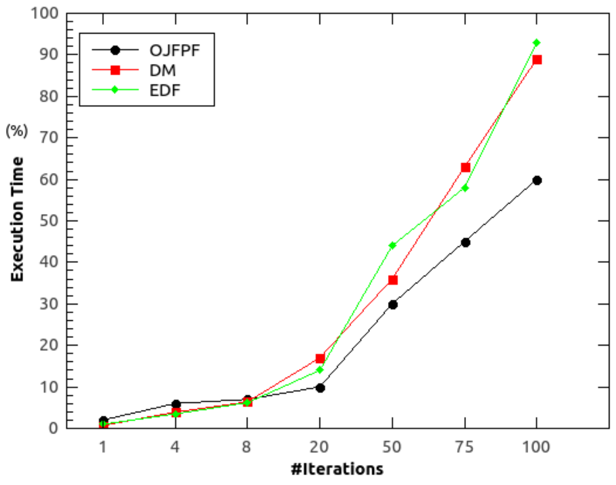

Fig. 5. The Trend of Execution Time

Fig. 6 shows the number of tasks shifts among the processors. It can be seen from the Fig. 6 that when the number of iterations are less than 8, there almost no shift exists. However, with the increase of iterations, the number of shifts are sharply increase for EDF and DM schedulings and the number of shifts increase very slow for OJFPF scheduling.

Table II shows the performance increases for our method OJFPF, in average cases of energy consumption, execution time as well as the number of reduced shifts, compared to the EDF and DM algorithms when the number of executing tasks range from 8 to 100.

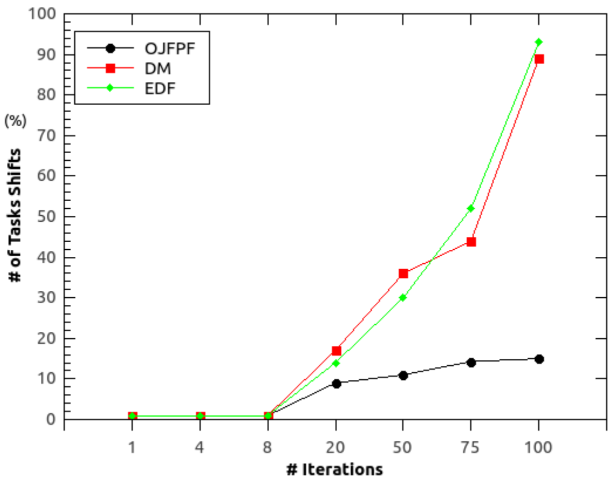

Fig. 6. The Trend of Number of Tasks Shifts

TABLE II
THE AVERAGE INCREASES(%) COMPARED TO EDF AND DM SCHEDULINGS

	Energy	Time	#Shifts
EDF	12%	16%	35%
DM	11.5%	15%	34.25%

V. CONCLUSION

This paper proposed a energy-aware real-time scheduling algorithm, OJFPF, which sends the evaluated optimal job to the fast processor in the heterogeneous multi-processor architectures. This algorithm uses both the hardware and software level parameters to assign the priorities to multiple tasks achieving an optimal assignment based on selected parameters. Extensive simulations on Loongson Multi-Processor architecture show the significant improvements in energy consumption and execution time as well as the reduced tasks shifts among processors.

REFERENCES

[1] Cong, Jason, and Bo Yuan. ”Energy-efficient scheduling on heterogeneous multi-core architectures.” Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design. ACM, 2012.
[2] Hill, Mark D., and Michael R. Marty. ”Amdahl’s Law in the Multicore Era.” IEEE Computer 41.7 (2008): 33-38.
[3] Kumar, Rakesh, et al. ”Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction.” Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. IEEE, 2003.
[4] Leung, Joseph Y-T., and Jennifer Whitehead. ”On the complexity of fixed-priority scheduling of periodic, real-time tasks.” Performance evaluation 2.4 (1982): 237-250.
[5] Liu, Chung Laung, and James W. Layland. ”Scheduling algorithms for multiprogramming in a hard-real-time environment.” Journal of the ACM (JACM) 20.1 (1973): 46-61.
[6] Dertouzos, Michael L., and Aloysius K. Mok. ”Multiprocessor online scheduling of hard-real-time tasks.” Software Engineering, IEEE Transactions on 15.12 (1989): 1497-1506.
[7] Awan, Muhammad Ali, and Stefan M. Petters. ”Energy-aware partitioning of tasks onto a heterogeneous multi-core platform.” Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th. IEEE, 2013.

[8] Davis, Robert I., and Marko Bertogna. ”Optimal Fixed Priority Scheduling with Deferred Pre-emption.” RTSS. 2012.

[9] Lakshminarayana, Nagesh, Sushma Rao, and Hyesoon Kim. ”Asymmetry aware scheduling algorithms for asymmetric multiprocessors.” (2008).

[10] Isci, Canturk, et al. ”An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget.” Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture. IEEE Computer Society, 2006.

[11] Wang, Gang, et al. ”Test and Repair Flow for Shared BISR in Asynchronous Multi-processors.” Asynchronous Circuits and Systems (ASYNC), 2014 20th IEEE International Symposium on. IEEE, 2014.

[12] Kopetz, Hermann. Real-time systems: design principles for distributed embedded applications. Springer, 2011.

[13] Zhu, Qi, et al. ”Optimizing extensibility in hard real-time distributed systems.” Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE. IEEE, 2009.

[14] Audsley, Neil C., et al. ”Real-Time scheduling: the deadline-monotonic approach.” in Proc. IEEE Workshop on Real-Time Operating Systems and Software. 1991.

[15] Dantzig, George B., and Mukund N. Thapa. Linear Programming 1: 1: Introduction. Vol. 1. Springer, 1997.

[16] Loongson-3B1500 Design Specification Version 1.5. Institute of Computing Technology, Chinese Academy of Sciences, July 2013.

[17] http://www.spec.org/cpu2006/publications/CPU2006benchmarks.pdf

[18] Luk, Chi-Keung, et al. ”Pin: building customized program analysis tools with dynamic instrumentation.” ACM Sigplan Notices 40.6 (2005): 190-200.

[19] R Tools: http://www.r-project.org/

[20] Baker, Theodore P. ”Multiprocessor EDF and deadline monotonic schedulability analysis.” 2013 IEEE 34th Real-Time Systems Symposium. IEEE Computer Society, 2003.