Real-Time Scheduling for Periodic Tasks in Homogeneous Multi-core System with Minimum Execution Time

Ying Li[1(✉)], Jianwei Niu[1], Jiong Zhang[1], Mohammed Atiquzzaman[2], and Xiang Long[1]

[1] State Key Laboratory of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, China
liying@buaa.edu.cn

[2] School of Computer Science, University of Oklahoma, Norman, OK 73019, USA

Abstract. Scheduling of tasks in multicore parallel architectures is challenging due to the execution time being a nondeterministic value. We propose a task-affinity real-time scheduling heuristics algorithm (TARTSH) for periodic and independent tasks in a homogeneous multicore system based on a Parallel Execution Time Graph (PETG) to minimize the execution time. The main contributions of the paper include: construction of a Task Affinity Sequence through real experiment, finding the best parallel execution pairs and scheduling sequence based on task affinity, providing an efficient method to distinguish memory-intensive and memory-unintensive task. For experimental evaluation of our algorithm, a homogeneous multicore platform called NewBeehive with private L1 Cache and sharable L2 Cache has been designed. Theoretical and experimental analysis indicates that it is better to allocate the memory-intensive task and memory-unintensive task for execution in parallel. The experimental results demonstrate that our algorithm can find the optimal solution among all the possible combinations. The Maximum improvement of our algorithm is 15.6%).

Keywords: Task affinity · Real-time scheduling · Periodic tasks · Homogeneous multicore system · Beehive

1 Introduction

With the changes of application, real time demands are being developed, e.g. scientific computing, industrial control and especially mobile clients. The popularity of mobile clients provided a broad space for the internet industry and presented higher demands on the performance of hardware. The traditional way to improve the processing speed relied on accelerating the clock speed, which resulted in a bottleneck due to a large amount of energy consumption. It forced companies to use multi-core technology [1–5]. But all of the traditional calculation models belong to Turing Machine which can only be used for serial instructions. If we wrote some parallel programmes on a single-core processor, they cannot be executed in parallel, essentially [6–9]. Therefore,

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 175–187, 2017.
DOI: 10.1007/978-3-319-59288-6_16

the single-core calculation models cannot be simply transplanted to multi-core. Parallel computing brings great challenges both to hardware structure and software design.

The *objective* of this paper is to find an efficient scheduling strategy which allows a set of real-time periodic and independent tasks to be executed in a **Homogeneous Multi-Core system** (HMC) with as little time as possible. In a multi-core system, the execution time of tasks is not a deterministic value and it is very difficult to find a sufficient condition for scheduling a set of periodic tasks. We solved this problem based on **task affinity** (defined in Sect. 3). First, we obtain the affinity between each task according to the actual measurement data. Second, we applied a scheduling heuristics algorithm to find an optimal parallel scheme and a reasonable execution sequence. This work will be useful to researchers for scheduling real-time tasks in a multicore processor system.

Real-time task scheduling for single-core processor was proposed in 1960 and the most representative algorithms are EDF and RM. Liu et al. [9–12] presented the scheduling policy and quantitative analysis of EDF and RM. In 1974, Horn proposed the necessary conditions for scheduling a set of periodic tasks. [13]. In 2005, Jiwei Lu [14] proposed a thread named Helper can be used to increase the percentage of Cache hits. But the time complexity of [14] algorithm is $O(N!)$ which had no practical significance. Kim, Chandra and Solihin studied the relationship between the fairness of sharing L2 Cache and the throughput of processor under the architecture of chip multiprocessors (CMP) and introduced some methods for measuring the fairness of sharing Cache [15]. Fedorova studied the causes of the unfairness of sharing Cache between tasks based on the SPEC CPU2000 [16]. Zhou et al. proposed a dynamic Cache allocation algorithm which can re-assign Cache resource by recording the parallel tasks' behaviors of using Cache [17]. Shao et al. [18] and Stigge et al. [19] divided the tasks into delay-sensitive ones and memory-intensive ones according to the characteristics of their memory access behaviors.

Although these works for multicore tasks scheduling have made some progress, most of them still used the same scheduling algorithms and analytic methods used in single-core processers, which indicated the execution time of a task is a deterministic value. But in multi-core system, the execution time is a nondeterministic value due to sharing of resources between tasks. Moreover, their experimental data is mostly obtained from simulation models which lack real data.

This paper is *different from previous work* in terms of using a nondeterministic scheduling algorithm for multicore processor and a real experimental environment.

In this paper, we *focus* on the scheduling strategy for a set of periodic and real-time tasks which can be executed on a multicore computing platform. We proposed a Task-Affinity Real-Time Scheduling Heuristics algorithm (TARTSH) for periodic tasks in multicore system based on a Parallel Execution Time Graph (PETG) which was obtained by accurately measuring the tasks' number of memory access and quantitatively analysing their delays due to resource competition. This algorithm focused on avoiding the execution of memory-intensive tasks in parallel, which can improve the real-time performance of the multi-core processor system.

The main *contributions* of this paper include:

- We proposed a quantitative method to measure the affinity between each task and obtained an affinity sequence according to the order of execution time which is affected by resource sharing.
- We designed a scheduling heuristic algorithm to find the best parallel execution pairs according to the task affinity and obtained an optimal tasks assignment method and scheduling strategy to minimize the sum of each core's execution time.

The rest of the paper is organized as follows. The Task Affinity model and related theorems are presented in Sect. 2. A motivational example is presented in Sect. 3 to illustrate the basic ideas of TARTSH algorithm. The multicore scheduling model is described in Sect. 4. The task-affinity real-time scheduling heuristics algorithm is presented in Sect. 5. The experimental results are presented in Sect. 6. Section 7 concludes the paper.

2 Basic Model

In this section, we introduce the Homogeneous Multi-Core system (HMC) architecture, followed by the Parallel Execution Time Graph (PETG) and definitions.

2.1 Hardware Model

In view of the research aim in this paper, we hope to find a multicore computing platform which can support a complete tool chains for writing a programme in advanced language and understanding the hardware program language for modifying hardware structure. Our investigation shows that Microsoft Research Beehive, which provides a multi-core prototype system, can meet our requirements. We modified the interconnection structure and storage architecture of Beehive by adding L2 Cache, clock interrupt, etc., to design a new multi-core processor, NewBeehive, as shown in Fig. 1.

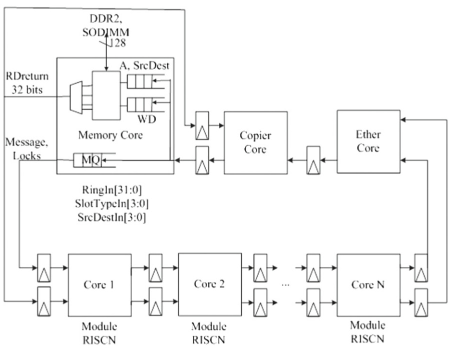

Fig. 1. The structure of NewBeehive.

NewBeehive is a RISC multi-core processor with bus architecture which can be implemented on FPGA. At present, NewBeehive can support up to 16 cores and each of them can be regarded as an independent computing entity. In Fig. 1, MemoryCore, CopierCore and EtherCore belong to service cores which are mainly designed to provide service for computing. MasterCore and Core1-Core4 belong to computing cores which are mainly used to execute tasks. In NewBeehive, Core1-Core4 are homogeneous and they share L2 cache and have their own private L1 Instruction Cache and L1 Data Cache. Core1-Core4 can access data from memory through L2 Cache, bus and MemoryCore. In order to meet the requirements of research, we incorporated some new functions in NewBeehive, including cache-coherent protocol, statistical analysis for Cache, clock interrupt and exclusive access to sharing resource, etc.

2.2 Definitions

In this paper, we use a *Parallel Execution Time Graph* (*PETG*) to model the tasks. The PETG is defined as follows:

Definition 2.1 ***Parallel Execution Time Graph* (*PETG*).** A *PETG G* = < *V*, *E* > is an undirected strongly connected graph where nodes V $= \{v_1, v_2, \ldots, v_i, \ldots, v_n\}$ represents a set of tasks and edges $E = \{e_{12}, \ldots, e_{ij}, \ldots, e_{nn}\}$ represents a set of execution time for which e_{ij} is the sum of the execution time of task v_i and the execution time of task v_j when they are executed in parallel, $e_{ij} = e_{ji}$, $i \neq j. e_{ij} = t^i_j + t^i_j$ where t^j_i is the parallel execution time of task v_i when it is executed in parallel with task v_j.

Each task's parallel execution time is recorded in the Task Parallel Execution Time Table which is used to calculate task affinity.

Definition 2.2 ***Task Parallel Execution Time Table* (*TPET*).** A *TPET A* is a table for which t^j_i represents the average parallel execution time of task v_i when it is executed in parallel with task v_j under different combinations of tasks and $t^j_i \neq t^j_i..t^j_i = \frac{\sum_{k=1}^{N} t^j_{i_k}}{N}$, where $N = C^n_m(v_i, v_j)$ indicates the number of different combinations of tasks including task v_i and v_j, N is the number of cores and m is the number of the tasks.

Task affinity which indicates the parallel appropriateness between tasks is recorded in the Task Affinity Sequence.

Definition 2.3 ***Task Affinity Sequence* (*TAS*).** A *TAS S* is an ordered sequence for which s_i represents the influence degree of task v_i affected by other tasks, $s_i = \{s^1_i, s^2_i, \ldots s^j_i, \ldots, s^n_i\}$, where $s^{j-1}_i \cdot \bar{s} < s^j_i \cdot \bar{s}$ and $i \neq j$. s^j_i is a tuple, $s^j_i = < v_j, \bar{s} >$, $s^j_i, \bar{s}$ is the difference ratio between the independent execution time and the parallel execution time of task v_i. $s^j_i \cdot \bar{s} = \frac{t^j_i - t_i}{t_i}$, where t_i represents the independent execution time of task v_i when it works on a single core and t^j_i represents the parallel execution time of task v_i when it is executed in parallel with task v_j.

Given a PETG *G*, TPET *A* and TAS *S*, the goal is to obtain a parallel execution set and a scheduling sequence on the target multicore computing platform *NewBeehive* to make the sum of each core's execution time as little as possible. To achieve this, our proposed methods need to solve the following problems:

- Task Affinity Sequence: Task affinity sequence is obtained by actually testing the independent execution time and the parallel execution time for each task on the multicore computing platform NewBeehive.
- Task Scheduling Sequence: Task scheduling sequence is composed of a tasks assignment which represents the best match of tasks work on different cores and an execution sequence which indicates the serial sequence of tasks work on one core.

3 Motivational Example

To illustrate the main techniques proposed in this paper, we give a motivational example.

3.1 Construct Task Affinity Sequence Table

In this paper, we assume all the real-time periodic tasks are independent so that and the execution time cannot be affected by the different combinations of tasks. The independent tasks we used in this paper are shown in Table 1. Tasks 1, 2, 3, 4, 5 and 6 are Matrix Multiplication, Heap Sort, Travelling Salesman Problem, Prime Solution, Read or Write Cache and 0-1 Knapsack Problem, respectively.

Table 1. Task list

Num	Tasks
v_1	Matrix
v_2	Sorter
v_3	Tsp
v_4	Prime
v_5	Cachebench
v_6	Pack

Table 2. Independent Execution Time (1000 clocks)

Num	Execution time on a single core				Average time
	Core1	Core2	Core3	Core4	
v_1	71619	72013	**72029**	71972	72015
v_2	74542	76712	74566	74510	75083
v_3	75317	78973	75317	75317	76231
v_4	75654	75654	75654	75654	75654
v_5	100641	100641	100637	100637	100639
v_6	72817	72816	72817	72816	72816

In order to calculate the delay between each task due to their sharing L2 Cache, we need to test the independent execution time TS*i* and parallel execution time TP*i* for each task, respectively. To make it easier to understand, we use two cores, Core3 and Core4 to execute the tasks in parallel.

First, we obtained the independent execution time TS*i* by executing task v_i on a single core which indicates task v_i can exclusively use all the resources and not be affected by other tasks. Table 2 is constructed by separately executing the target tasks on a single core of NewBeehive. For the better result, we take the average of four tests. Table 2 shows one task's respective execution times on different cores are basically the same, which indicates Core1 $\sim$ Core4 are homogeneous. And it accords well with the design of NewBeehive in Sect. 2.

Second, we test the parallel execution time Tp*i* by executing task v_i on one core and other tasks on the left cores. These tasks will be affected by each other due to sharing L2 Cache. The value $t_{v1}^{v2} = 76062$, which represents the parallel execution time of task v_1 when it works on Core3 and v_2 works on Core4 at the same time. And $t_{v2}^{v1} = 83811$ represents the parallel execution time of task v_2. They are different because they belong to different tasks' parallel execution time.

According to Table 2, we find each task's parallel execution time is longer than its independent execution time. Furthermore, if a task belongs to the memory-intensive application, it will significantly increase the other task's execution time. For example, task 5 is a Cachebench, which accesses data from memory frequently and all the other tasks will have a great delay when they are executed in parallel. In Table 2, task 1's independent execution time on core3 is 72029, but its parallel execution time on core3 is 90644 when task 5 works on core4.

Third, we calculated the influence ratio between each task based on its independent execution time and parallel execution time, as shown in Table 3. E.g., $= \frac{t_{v2}^{v1} - t_{v2}}{t_{v2}} = \frac{83811-74542}{74542} = 12.4\%$.

By analyzing the task affinity sequence s_i in Table 3, we conclude the following two results:

(1) In a row, if the task affinity grows very little, it indicates the task in this row belongs to memory-unintensive application. The reason is the task's parallel execution time is less influenced by other tasks when it rarely accesses memory, e.g. task 4.

(2) In a column, if the task has a significant impact on other tasks, it indicates the task in this column belongs to memory-intensive application. The reason is the task will severely impact the execution time of others when it frequently updates L2 Cache and uses Bus, e.g. task 5.

Table 3. Influence Ratio of Two Cores (Unit: %)

Cores	Core4						
Core3		v_1	v_2	v_3	v_4	v_5	v_6
	1	–	5.6	0.8	0.3	25.9	4.0
	2	**12.4**	–	2.7	1.7	65.4	8.3
	3	2.5	2	–	0.01	21.3	0.2
	4	0.24	0.22	0.11	–	0.6	0.12
	5	27.4	26.3	8.3	3.6	–	21.7
	6	14.6	11.2	0.3	0.01	55.7	–

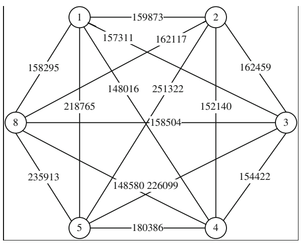

Fig. 2. Parallel execution time graph.

3.2 Find an Optimal Tasks Scheduling

In order to find an optimal Task Scheduling Sequence, we apply a task-affinity real-time scheduling heuristics algorithm (TARTSH) based on graph theory to assign tasks. According to the conclusions in Sect. 3, it is better to allocate the memory-intensive task and memory-unintensive task to be executed in parallel, which can reduce the competition for resources and improve the real-time performance.

First, we draw a Parallel Execution Time Graph (PETG) based on Table 2, as shown in Fig. 2. Each edge in graph G is the sum of the parallel execution times of two nodes, e.g. $e_{12} = t_1^2 + t_2^1 = 76062 + 83811 = 159873$.

Second, we find the best parallel execution pairs based on the TARTSH algorithm. We obtained a global task affinity sequence by ordering each task's parallel influence. The parallel influence of task v_i indicates the total influence of task v_i to all the other tasks when they are executed in parallel, which is calculated by adding all the $s_j^i.\bar{s}$, where i = 1,2,…,n and i $\neq$ j. For example, according to Table 3, the parallel influence of task $v_5 = 25.9 + 65.4 + 21.3 + 0.6 + 55.7 = 168.9$ and the global task affinity sequence (GTAS) is $\{v_5, v_1, v_2, v_6, v_3, v_4\}$. And the best parallel execution pairs are obtained by finding their best match task which has the strongest affinity according to the order the global task affinity sequence. E.g., $\{<v_5, v_4>, <v_1, v_3>, <v_2, v_6\}$.

Third, we find the optimal task scheduling sequence by allocating the tasks in each sub-sequence in the global task affinity sequence to their appropriate cores based on the task affinity sequence of the most influence task. In this paper, the most influence task is task v_5 which indicates it has the largest influence on the other tasks. And the task affinity sequence of task v_5 is $\{v_4, v_3, v_6, v_2, v_1\}$. Therefore, the optimal task scheduling sequence is composed of the task execution sequence on each core. P(c_i) is the set of tasks assigned to core c_i. E.g., $P(c_3) = \{v_5, v_1, v_2\}$ and $P(c_4) = \{v_4, v_3, v_6\}$. If two tasks have the same index in the different cores, they will be executed in parallel, e.g. v_1 is executed with v_3.

4 Multicore Scheduling Model

In this section, we propose a multicore scheduling model to achieve an optimal tasks assignment method and scheduling strategy in *HMC* system that makes the sum of each core's execution time as little as possible. First, the notations and assumptions used to construct the multicore scheduling model are presented in Table 4. Then, the theorems are introduced.

The aim of multicore scheduling model is to minimize the total execution time on the condition that the set of periodic and independent tasks can be scheduled. The total execution time is defined as:

$$\begin{aligned} T_{opt}(V) &= \min(\sum\nolimits_{c_i \in C} T(c_i)) \\ &= \min(\sum\nolimits_{v_i \in V} TP(v_i) + \sum\nolimits_{v_i \in V} TD(v_i)) \end{aligned} \quad (1)$$

Table 4. Notations of TARTSH Algorithm

V	A set of periodic and independent tasks	V	A set of periodic and independent tasks
T_{opt} (P)	The optimal tasks scheduling with the minimum execution time	$\beta(v_i)$	The parallel influence of task v_i to all the other tasks
$TA_{opt}(S)$	The optimal tasks assignment with the minimum sum of task affinity	$\theta(v_i)$	The parallel influence of the best match tasks $\mathrm{M}(v_i)$ to task v_i
$\mathrm{M}(v_i)$	the best match tasks of task v_i	$T(c_i)$	the execution time of core c_i
$\bar{V}_i$	The set of tasks in $\mathrm{M}(v_i)$	$TS(v_i)$	The independent execution time of task v_i
$\mathrm{P}(c_i)$	The task execution sequence assigned to core c_i	$TP(v_i)$	The parallel execution time of v_i
$\varepsilon(v_i)$	The parallel influence of all the other tasks to task v_i	$TD(v_i)$	The delay when task v_i is executed in parallel

Where $TD(v_i)$ is defined as:

$$TP(v_i) = TS(v_i) \times (1 + \theta(v_i)) \tag{2}$$

$$TD(v_i) = TS(v_i) \times (1 + \varepsilon(v_i)) \tag{3}$$

Then, according to Eqs. (1)–(3), it holds that

$$T_{opt}(V) = \min\{\sum\nolimits_{v_i \in V} [TS(v_i) \times (2 + \theta(v_i) + \varepsilon(v_i))]\} \tag{4}$$

Theorem 4.1. If a set of periodic and independent tasks are executed in parallel, the optimal tasks assignment TA_{opt} composed of $\mathrm{M}(v_i)$ can be obtained by sorting its $\beta(v_i)$ in ascending order.

Proof: According to the definition,

$$\begin{aligned} TA_{opt}(S) &= \{S_1, \ldots, S_m, \ldots, S_n\} \\ &= \sum\nolimits_{m=1}^{N} S_m \cdot s \end{aligned}$$

where, $S_m = M(v_i)$, $S_m \cdot s = \sum_{v_i, v_j \in S_m} (s_i^j \cdot \bar{s} + s_i^j \cdot \bar{s})$ (defined in Sect. 3), and N is the number of the cores. Then,

$TA_{opt}(S) = M^1(v_l), \ldots, M^m(v_i), \ldots, M^n(v_j)$, where $\bar{V}_l \cup \ldots \bar{V}_i \cup \ldots \cup \bar{V}_j = V$, $\bar{V}_i \cap \bar{V}_j = \emptyset$ and $\beta(M^{m-1}(v_k)) > \beta(M^m(v_i))$.

Assume $\beta(M^{m-1}(v_k)) < \beta(M^m(v_i))$, then there is a new the optimal tasks assignment TA'_{opt} whose total task affinity is smaller than TA_{opt}'s. It holds that

$$TA'_{opt}(S') = \{S'_1, \ldots, S'_m, \ldots, S'_n\}$$
$$= \sum\nolimits_{m=1}^{N} S'_m \cdot s$$

where $S'_m \cdot s = \sum_{v_k, v_l \in S'_m} (s'^{l}_{k} \cdot \bar{s} + s'^{k}_{l} \cdot \bar{s})$

If $\beta(M^{m-1}(v_k)) < \beta(M^{m}(v_i))$, then

$\sum_{v_k, v_l \in S'_m} (s'^{l}_{k} \cdot \bar{s} + s'^{k}_{l} \cdot \bar{s}) > \sum_{v_i, v_j \in S_m} (s^{j}_{i} \cdot \bar{s} + s^{j}_{i} \cdot \bar{s})$ which indicates $\sum_{m=1}^{N} S'_m \cdot s > \sum_{m=1}^{N} S'_m \cdot s$

And it is different from assuming which indicates $\mathrm{M}(v_i)$ in $TA_{opt}(S)$ is ordered by its $\beta(v_i)$.

Theorem 4.2. Based on $TA_{opt}(P)$, the optimal tasks scheduling T_{opt} can be obtained by making the tasks executed with their strong affinity tasks.

Proof: Assume the most influence task with the largest $\beta(v_i)$ is v_{max}, and its task affinity sequence $s_i(v_{max}) = \{s^{1}_{max}, \ldots s^{j}_{max}, \ldots, s^{n}_{max}\}$ (defined in Sect. 3). Then,

$TA_{opt}(P) = \{\mathrm{P}(c_1), \ldots, \mathrm{P}(c_m), \ldots, \mathrm{P}(c_N)\}$ and $\mathrm{P}(c_m) = \ < v^{1}_{m}, \ldots, v^{k}_{m}, \ldots, v^{n}_{m} >$, where the tasks in $\mathrm{P}(c_m)$ are the same with those in S_m but ordered according to the task affinity of v_{max} from small to large.

Assume a task v' is assigned to core c_m to replace the task v^{k}_{m} and $\theta(v_{max}, v') > \theta(v_{max}, v^{k}_{m})$. Then, a new optimal tasks scheduling $TA'_{opt}(P')$ is obtained.

$TA'_{opt}(P') = \{\mathrm{P}'(c_1), \ldots, \mathrm{P}'(c_m), \ldots, \mathrm{P}'(c_N)\}$ and $\mathrm{P}'(c_m) = \ < v^{1}_{m}, \ldots, v^{k}_{i}, \ldots, v^{n}_{m} >$. According to the Eqs. (2), we have that

$$TP(v_{max}, v') = TS(v_i) \times (1 + \theta(v_{max}, v'))$$

Therefore, $TP(v_{max}, v') > TP\left(v_{max}, v^{k}_{m}\right)$ which indicate

$$TA'_{opt}(P') > TA'_{opt}(P)$$

And it is different from assuming.

5 TARTSH Algorithm

In this section, we propose a task-affinity real-time scheduling heuristics algorithm (*TARTSH*) to find the T_{opt} which has the minimum total execution time on the condition that the set of periodic and independent tasks can be scheduled in a given HMC according to task affinity.

Algorithm 5.1 shows the *TARTSH* algorithm. Initially, we build a matrix $\mathrm{TA}[V_n][V_n]$ to record task affinity between tasks and $\mathrm{TA}[v_i][v_j]$ represents the $s^{j}_{i} \cdot \bar{S}$ (defined in Sect. 3). The variables $S(v_i)$, C_i(S) and PS are used to record the parallel influence of task v_i, the already assigned tasks on the core c_i and the global priority of all the tasks based on the task affinity, respectively. And $\mathrm{U}(Ci)$ is a function to calculate the resource utilization rate of core *Ci* and *Li*(n) is the least upper bound of the utilization ratio of core c_i.

The *TARTSH* algorithm tries to find the best parallel execution sequence according to the task affinity and obtained an optimal tasks assignment method and scheduling strategy to make the sum of each core's execution time as little as possible. From line 4 to line 16, the algorithm construct the priority of each task, $PS[V_n][V_n]$, which satisfies the condition $PS[v_i][v_x] > PS[v_i][v_y]$, where $x < y$, $PS[v_i][v_x]$ is the task affinity between v_i and v_x. Then, we sort $PS[V_n][V_n]$ based on $PS[V_n][0]$ in line 17. From line 19 to line 23, the task pairs with the highest tasks affinity will be assigned to the empty cores. *PS'* is obtained in line 24 by deleting the assigned tasks from *PS*. From line 25 to line 38, the tasks assignment on each core is obtained by finding the best match task for the core's latest task based on task affinity.

Algorithm 5.1. Task-Affinity Real-Time Scheduling Heuristics Algorithm (TARTSH)

Input: (1) An Independent Execution Time A; (2) A Parallel Execution Time B; (3) A graph model PETG G =<*V, E*> (4) A Homogeneous multi-core system Pm;
Output: The optimal tasks scheduling with the minimum total execution time

1:Initialize a $N \times N$ matrix TA[V_n][V_n]$\leftarrow$ 0;
2:Initialize *S*(*V*) $\leftarrow$ 0, *C*(*S*) $\leftarrow$ 0, *PS*[V_n][V_n]$\leftarrow$ 0, *k* = 0;
3: Initialize *CoreNum* = *M*;
4:**for** $v_i \in V$ **do**// construct *PS*[*N*][*N*]
5: *k* = *N*
6: **while** *k* > 0 **do**
7: ta = 0
8: v_j = the task of TA[V_i][*k*]
9: **for** $v_j \in V$ **do**
10: **if** TA [v_i][v_j] > ta **then**
11: ta = TA [v_i][v_j]
12: *PS*[v_i]$\leftarrow$ v_j//find the largest task affinity of v_i
13: *k* = *k* – 1
14: **end for**
15: **end while**
16:**end for**
17:*PS* = Sort(*PS*[v_i])
18:*m* = 0
19:**while** *m* < *CoreNum* **do** // assign tasks to the empty cores
20: v_m = the task of *PS*[*m*], v_n = the task of *PS*[v_m] [*0*]
21: *Ci*(*S*) $\leftarrow$ <v_m, v_n>
22: *m* = m + 1
23:**end while**
24: *PS'* = *PS* - $\sum_{i=0}^{M} Ci(S)$
25:**for** *Ci* in *Pm* **do**
26: v_c = find the latest task in *Ci*(*S*)
27: *PS''*[v_c] = *PS'*[v_c]
28: **while True do**
29: v_p = max(*PS''*[v_c][v])
30: U(*Ci*) ≡ U(*Ci*) ∓ U(v_p)
31: **if** U(*Ci*) ⩽ Li(*n*) **then**
32: *Ci*(*S*) $\leftarrow$ v_p
14: **end for**
15: **end while**
16:**end for**
33: *PS''*[v_c] = *PS''*[v_c] - v_p
34: *PS'*[v_c] = *PS''* [v_c]
35: **break**
36: **else**:
37: *PS''*[v_c] = *PS''*[v_c] - v_p
38:**end for**

6 Experiments

Experimental results are presented in this section. To demonstrate the effect of the *TARTSH* algorithm across different cores, we complete our experiment in a homogenous multi-core system with 2 cores, 4 cores and 8 cores, respectively. Our main method is to generate all the periodic tasks sets consisted of real-time tasks defined in Table 1 based on random algorithm and record their execution time, cache read failure times and hit rate, respectively. Then, the effectiveness of the *TARTSH* algorithm is proved according to the statistical data.

6.1 Periodic Tasks Set

We design different sizes of periodic tasks set consisted of different real-time tasks defined in Table 1 by making them executed randomly for many times, as shown in Table 5. In our experiment, the number of periodic tasks set is limited between 100 and 1500 for very small number of tasks will lead to inaccurate, but a large number of tasks will increase the difficulty of collecting data. The execution sequence of tasks is also generated randomly. E.g., Set1 just includes two tasks and they will be {T1, T2} or {T1, T3} or {T1, T4} or other combinations of two tasks. And we execute them for 50 times to obtain a periodic tasks set with 100 tasks, e.g., {{T1, T2}, {T1, T2},…, {T1, T2}}.

Table 5. Periodic tasks set table

Set No.	Number of tasks	Size of set	Number of cores
Set1	2	100	{1,2}
Set2	4	500	{1,2,3,4}
Set3	6	1000	{1,2,3,4,5,6}
Set4	8	1500	{1,2,3,4,5,6,7,8}

6.2 Task Affinity

In this paper, our purpose is to schedule a set of real-time periodic and independent tasks with as little time as possible based on the task affinity. Task affinity can be measured qualitatively based on the parameters of cache read-failure times, task execution time, etc., which are obtained by executing the periodic tasks sets in different size of homogenous multi-core systems, as shown in Table 6.

In advance, we know T1, T3 and T5 access memory frequently and T2 and T4rarely access memory. Table 6 shows a part of the statistical data of set1 and it indicates T1 and T5 have the strongest affinity for they share data. But T1 and T3 will cause the failure of reading cache for their data is stored on different lines of cache.

Table 6. A part of statistical data of set1

No.	Tasks	Cache performance parameters on one core			Cache performance parameters on two core		
		Read times	Read-failure times	Hit Rate	Read times	Read-failure times	Hit Rate
1	T1, T2	725301	54397	92.5	401631	34942	91.3
2	T1, T5	638120	15953	97.5	309162	9893	96.8
3	T2, T4	65390	6931	89.4	39125	3717	90.5
4	T2, T5	640145	60174	90.6	392174	41178	89.5
5	T1, T3	1025471	255342	75.1	8946756	1261493	85.9
6	T2, T3	825301	179090	78.3	579834	92–93	83.6

7 Conclusion

In this paper, we propose a task-affinity real-time scheduling heuristics algorithm (TARTSH) for periodic and independent tasks in a homogeneous multicore system based on a Parallel Execution Time Graph (PETG) to minimize the execution time. We build multicore scheduling model to obtain the best parallel execution pairs and scheduling sequence based on task affinity. The experimental results show that TARTSH algorithm spends less time than any other combination which is implemented in a real homogeneous multicore platform.

Acknowledgments. This work was supported by the National Natural Science Foundation of China (61572060, 61190125, 61472024), 973 Program (2013CB035503), and CERNET Innovation Project 2015 (NGII20151004).

References

1. Bastoni, A., Brandenburg, B.B., Anderson, J.H.: An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers. In: Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS), pp. 14–24 (2010)
2. Liu, J.W.S.: Real-Time System. Pearson Education (2002)
3. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real time environment. J. ACM **20**(1), 46–61 (1973)
4. Davari, S., Dhall, S.K.: An online algorithm for real-time tasks allocation. In: IEEE Real-time Systems Symposium, pp. 194–200 (l 986)
5. Baruah, S.K., Li, H., Stougie, L.: Towards the design of certifiable mixed-criticality systems. In: The Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 13–22 (2010)
6. Lauzac, S., Melhem, R., Mosse, D.: Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In: 10th Euromicro Workshop on Real Time Systems, pp. 188–195, June 1998
7. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. **4**, 1–44 (2011)

8. Mok, A.K.: Fundamental design problems of distributed systems for the hard real-time environment. Ph.D. Dissertation, MIT (1983)
9. Lakshmanan, K., de Niz, D., Rajkumar, R., Moreno, G.: Resource allocation in distributed mixed-criticality cyber-physical systems. In: The 30th International Conference on Distributed Computing Systems (ICDCS), pp. 169–178 (2010)
10. De Niz, D., Lakshmanan, K., Rajkumar, R.: On the scheduling of mixed-criticality real-time task sets. In: The 30th Real-Time Systems Symposium (RTSS), pp. 291–300 (2009)
11. Guan, N., Ekberg, P., Stigge, M., Yi, W.: Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In: The 32rd Real-Time Systems Symposium (RTSS), pp. 13–23 (2011)
12. Burchard, A., Liebeherr, J., Oh, Y.F., Son, S.H.: New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. on Comput. **44**(12), 1429–1442 (1995)
13. Han, C.C., Tyan, H.: A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In: The 18th IEEE Real-Time Systems Symposium, San Francisco, pp. 36–45 (1997)
14. Lu, J., Das, A., et al.: Dynamic helper threaded prefetching on the sun ultra SPARC CMP processor. In: The 38th Microarchitecture, pp. 93–104, October 2005
15. Kim, S., Chandra, D., Solihin, Y.: Fair cache sharing and partitioning in a chip multiprocessor architecture. In: 13th International Conference on Parallel Architecture and Compilation Techniques, Los Alamitos, CA, pp. 111–122 (2004)
16. Fedorova, A.: Operating System Scheduling for Chip Multithreaded Processors. Ph.D. thesis, Harvard University (2006)
17. Benhai, Z., Jianzhong, Q., Shukuan, L.: Dynamic shared cache allocation algorithm for multicore professor. J. Northeast. Univ. **32**(1), 44–47 (2011)
18. Shao, J., Davis, T.: A burst scheduling access reordering mechanism. In: 13th International Symposium on High Performance Computer Architecture, pp. 285–294 (2007)
19. Stigge, M., Ekberg, P., Guan, N., et al.: On the tractability of digraph-based task models. In: 23rd Euromicro Conference on Real-Time Systems (ECRTS), Porto, Portugal, pp. 162–171 (2011)