A Novel PSO based Task Scheduling Algorithm for Multi-core Systems

Jia Tian1,2, Wei Hu1,2,\*, Yonghao Wang3, Lin Li1,2, Peng Ke1,2, Kai Zhang1,2

1College of Computer Science and Technology, Wuhan University of Science and Technology, China

2Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, China

3The DMT Lab, Birmingham City University, UK

springreef@foxmail.com, \*huwei@wust.edu.cn, lilin@wust.edu.cn, ke\_peng@wust.edu.cn, zhangkai@wust.edu.cn

*Abstract*—Multi-core processors have been the mainstream in computer architecture. It also provides the enhancement of the parallelism degree of multiple tasks. An emerged challenge is how to schedule the multiple tasks to the cores for high efficiency. In this paper, a novel task scheduling algorithm is proposed for multi-core systems. This algorithm is based on optimized particle swarm algorithm, which is used to find the optimal solution for the task scheduling. The experimental results have showed that the proposed algorithm can improve the efficiency of task scheduling for multi-core systems.

*Keywords*—Particle swarm optimization; multi-core; multi-thread; task scheduling

# Introduction

Large number of transistors are integrated onto the single chip, which provides huge potential to improve the performance of the processors [1]. When CPU with a single core confronts with the great challenge in system performance and power-consuming, multi-core processors have been taken as a promising solution [2-4]. More cores on a single chip can improve the performance and cut down the power consumption by reducing the frequency of each core. Multiple tasks can be executed in parallel on multiple cores. The high parallelism of the tasks enhances the performance of the systems. However, a new challenge is emerging for multi-core processors [5]. When more tasks are running in parallel, they are likely to communicate with other cores. It plays an important role in the system performance on how to schedule these tasks to the different cores. The scheduler should consider the performance of the multiple cores while making the scheduling decision. When the tasks are scheduled to different cores, the communication relationship can vary. The key is to reduce the communication penalty, scheduling time and improve the performance of the multi-core processors [6].

Recent works have proposed to address this challenge of the scheduling schemes for multi-core processors. Partitioned hierarchical real-time scheduling was proposed for multi-core processors [7]. In this design, the applications are considered as well-defined components in a hierarchical manner. The scheduling could be operated in the different hierarchies. Preemptibility-aware scheduling (PAS) was proposed as a responsive scheduling algorithm to reduce the scheduling latency in multi-core systems [8]. One core prepared for the urgent interrupt by both of interrupt-enabled and being in preemptible sections. And other approaches were also proposed to schedule the tasks for high performance. In this paper, a PSO (Particle Swarm Optimization) based task scheduling algorithm is proposed for multi-core processors. The optimized particle swarm algorithm is used to find the optimal solution for the task scheduling. It provides a novel approach to improve the performance of multi-core processors.

This paper is organized as the follows. Section 2 introduces the related works. Section 3 describes the system model and the PSO based task scheduling algorithm. The experiments and result analysis are discussed in Section 4. And at last, we give the conclusions in Section 5.

# Related Work

When there is only a single core on chip, the scheduling algorithm aims to manage this core with high efficiency. The system cannot have the parallelism in thread level. No threads can share this core at the same time. However, the tasks can run in parallel on multiple cores on the same chip. The communication among the tasks will consume more time. How to map the tasks to the cores is one of the key element to solve the above problem. There are existing works on this problem to provide better solutions [9-12].

There are different types of scheduling algorithms. When multiple tasks are mapped to the multiple cores, the efficiency is the main target. Static scheduling algorithms were proposed to complete the mapping between the tasks and the cores. Heuristic mapping algorithm [13], genetic mapping algorithm [14], QoS guaranteed mapping algorithm [15] and multi-target mapping method for mesh network [16] are such typical algorithms. The scheduling was determined before the execution of the tasks. It means that such algorithms could obtain a well optimization of the scheduling. However, the scheduling could not be adjusted at run-time. When the situations were changed, the algorithms had to re-calculate the scheduling approach. Dynamic scheduling algorithms could manage the scheduling of tasks at run-time according to the instant proofing [17-20]. Such algorithms could reduce the traffic on-chip. Some specific hardware or software units could be added for the efficient management. When the traffic was heavy, these units were the bottleneck. A different algorithm is proposed in [21], which is a scenario based mapping method. The scenarios is set as the states in a state machine. When the states changed, the scheduling would be triggered.

This work was supported by National Natural Science Foundation of China under granted No. 61403287 and Natural Science Foundation of Hubei Province under granted No. 2014CFB288.

When more cores are available, many algorithms focuses on both of the scheduling and the power consumption. DVS/DVFS (Dynamic Voltage Scaling/Dynamic Voltage Frequency Scaling) are traditionally used on single core chip. Now they are also used in multi-core processors for both efficiency and power consumption [22-24]. [25] focused on how to detect the idle time of the cores to put the shared memory into sleep to save energy. [26] proposed an approach for task scheduling based on the run-time characteristics of individual tasks, which were considered as the critical factor in making decisions for the scheduling. [27] presented the concept that the tasks with the same run-time features could be dispatched a clustered region of the cores. Such algorithms could provide new designs to take into account the efficiency and energy consumption. However, it also means the compromise of the efficiency.

The task scheduling itself should be improved to find a fast approach to complete the task dispatching. The potential optimal scheduling sequence should be provided to the system for the improvement of the performance. In this paper, PSO is used for this target.

# PSO Based Task Scheduling Algorithm

## System Model

The technical advantages of multi-core processor is to support the multiple tasks running in parallel on multiple cores. It is parallelism in thread level. The cores have relatively simple structure. The tasks will communication with each other through the bus, wires between cores or shared memory. A system model is proposed to abstract and simplify the multi-core systems.

For a multiprocessor system, assuming that there are *n* identical processors. The number of tasks to be scheduled is *m*. The directed acyclic graph (DAG) is used to represent the tasks in the system. DAG can be represented by quintuple G =(V, E, R, C, W).



1. DAG with direction and edge weights

V = {Vi} is the set of vertices, which is the collection of the tasks. Vertex Vi is used to represent a task. (Vi, Vj) is used to represent the edge between the two vertices Vi and Vj.

E = {e} is the set of the edges. e(i, j) is the edge between the two vertices Vi and Vj, i.e. (Vi, Vj). All edges in E are directed ones.

R = {Ri} is the set of execution time of the tasks. Ri is used to represent the execution time of a task Vi.

C = {c} is the set of the weight of the edges. The weight is a compound factor, which is the coupling degree of the traffic and control between the tasks.

W = {w(Vi, Vj} is the set of communication overhead between the tasks Vi and Vj. W is an additional information of the edges. When the two tasks are running on the same core, their communication overhead is at the minimum and the corresponding w is also at the minimum.

Fig. 1 shows the DAG with ten tasks. The values of the edges are the overhead of communication between tasks. They are just the weights of the edges.

## Algorithm Design

PSO algorithm is proposed for the simulation of a simple social model [28]. In the PSO algorithm, each optimization problem of potential solutions is search in the space of a bird, known as "particles". All particles have an adaptive value (Value Fitness) determined by the optimized function, and each particle has a velocity that determines the direction of their flight and the displacement of each step. Then the particles follow the current optimal particle in the solution space. PSO algorithm needs to initialize a group of random particles (random solution), and then to find the optimal solution through the iteration. In each iteration, the particle tracks two "extreme" to update them. The first one is the optimal solution of the particle itself, which is called the individual extremum. The other is the optimal solution for the whole population, which is called the global extremum. The two basic formulas of particle swarm optimization algorithm are as follows.

$v\_{ij}^{k+1}=w\*v\_{ij}^{k}+c\_{1}\*r\_{1}\*\left(pBest\_{ij}^{k}-x\_{ij}^{k}\right)+c\_{2}\*r\_{2}\*(gBest\_{ij}^{k}-x\_{ij}^{k})$ (1)

$x\_{ij}^{k+1}=x\_{ij}^{k}+v\_{ij}^{k+1}$ (2)

In (1) and (2), the particle velocity is $v\_{ij}^{k+1}$; w is the inertia weight, and $ x\_{ij}^{k}$ is the position of the particle. $ pBest\_{ij}^{k}$ and $gBest\_{ij}^{k}$ are defined as optimal solutions of the particle itself and the whole group respectively. $r\_{1}$ and $r\_{2}$ is random values (0 or 1). $c\_{1}$ and $c\_{2}$ is the learning factors. k is the number of iterations; i is the number of particles; and j is the dimension of the target space.

This algorithm is designed for multi-core processors with multiple tasks. In this optimization, the positions of the particles are represented by X which is a binary value:

$X=\left[\begin{matrix}X\_{11}&\cdots &X\_{1n}\\\vdots &\ddots &\vdots \\X\_{m1}&\cdots &X\_{mn}\end{matrix}\right]$ (3)

Each Xij in X is 0 or 1. Xij = 1 means task i is dispatched to core j. If a row in the matrix X is zero, it means this task can be dispatched to any core.

The speed of the particles S can be normalized as:

$S=\left[\begin{matrix}S\_{11}&\cdots &S\_{1n}\\\vdots &\ddots &\vdots \\S\_{m1}&\cdots &S\_{mn}\end{matrix}\right]$ (4)

At this time, the original location in (2) are unable for the situations. After the initialization, the algorithm starts to search for the optimal solution. It goes through the V matrix of each line, to find the maximum value of the location of each row, and the location of the $x\_{ij}$ is set to 1. And then, the following (5) is used to replace the position in (2):

 $if\left(rand()<sig(s\_{ij})\right) x\_{ij}=1 else x\_{ij}=0$ (5)

In (5), rand () is a random number between [0,1], sig($s\_{ij}$) is the Sigmoid function shown as follows:

$sig\left(s\_{ij}\right)=1/(1+e^{-s\_{ij}})$ (6)

In order to ensure that the group can move evenly, the speed range of particles is set as [-4,4]; and the $s(v\_{ij})$’s range is closer to the middle value, rather than near from 0 to 1.

The weight w in (1) has a great influence on the global search ability and local search ability of the algorithm. When w is very small, the PSO algorithm is easy to fall into local extreme value. When the w is large, the convergence of PSO algorithm is low. So the weight w range is set in [0.6, 1.0]. In this range, the balance of the algorithm is suitable, and it is easy to find the global optima through the iterations.

The weight w is got by using the linear decreasing:

$w=w\_{max}-\frac{w\_{max}-w\_{min}}{k\_{max}}\*k$ (7)

In (7), $w\_{max}$ is the maximum weight; $w\_{min}$ is the minimum weight; $k\_{max}$ is the number of iterations of the algorithm; k is the number of iterations of the algorithm. Such approach can improve the convergence rate of the algorithm, and provide significant improvement in finding the optimal solution.

After each update of the position, the fitness values will be re-computed. The edge sets are traversed. If two tasks are scheduled to the same core, the adaptive value is set to 1, which is the minimum value. The communication overhead is represented by $F(i)$, which the communication time of each pair of tasks. The number of edges at system initialization is represented by num. Then $F(i)$ can be calculated as follows:

$F(i)=\left\{\begin{array}{c}b\*w\left(i\right)\*c\left(i\right) ,different cores.\\b\*c\left(i\right)\*1 ,on the same core.\end{array}\right.$ (8)

In (8), b is the coefficient of the scheduling time; w (i) is the weight on the edge of the DAG; c(i) is the data control correlation between tasks. The fitness function of the system is:

$f\left(s\right)=\sum\_{i=1}^{num}F\left(i\right)$ (9)

The flow of discrete particle swarm algorithm is as follows:

Step 1. In this step, the following works will be completed including the initialization of the original DAG, the initialization of the particle position and velocity, and setting the number of cycles t =0.

Step 2. In this step, the particles' fitness values are calculated, and the pBest and gBest are set up.

Step 3. In this step, the particles’ velocity V and position X are updated.

Step 4. In this step, the particles’ fitness values are calculated, the pBest and gBest are setup, and t = t + 1.

Step 5. If the maximum cycle times are completed, the optimal solution is output; otherwise the algorithm goes to step 3.

The diagram of this algorithm is shown in Fig. 2 as the follows:



1. Algorithm Flow Diagram

# Experimental Results and Analysis

In order to obtain the efficiency of the algorithm, the experiments are set up with a series of random characteristic parameters of the tasks to generate the DAG. The generated DAG is used as the input set of the algorithm. The population size and iteration parameters are set up for the experiments.

For each given number of on-chip cores, 100 DAG are generated randomly. The genetic algorithm based task scheduling is used for the comparison. The setup of the experimental environment is based on the system with Intel(R) Core(TM) i3-3240 CPU(3.40GHz), 8GB main memory and Windows 7 Professional operating system. The tests are implemented by MATLAB 7.0.

The main parameters of the algorithm are: particle population size is 30; learning factors c1 and c2 are both 2; the maximum iteration number is 100; the maximum velocity is 4; w(i) and c(i) are 1 to 10; b is set as the natural coefficient e.

**Table 1 Comparison of simulation results**

|  |  |  |
| --- | --- | --- |
| Number of processor cores | Number of threads | The shortest time to obtain the global optimal solution (ms) |
| DPSO | GA |
| 4 | 30 | 39 | 48 |
| 60 | 71 | 81 |
| 80 | 66 | 97 |
| 6 | 30 | 32 | 40 |
| 60 | 58 | 73 |
| 80 | 61 | 89 |
| 8 | 30 | 26 | 31 |
| 60 | 52 | 61 |
| 80 | 59 | 82 |



1. Particle swarm optimization algorithm curve

As Table I shows, the efficiency of the algorithm will be improved when the number of cores increases. However the amplitude of the improvement is reduced until the algorithm is stable. When the number of tasks increases, the efficiency of the algorithm is gradually improved. Fig. 3 shows another advantage of this algorithm. This algorithm is stable in the iteration number of 27 times, and this PSO based algorithm can find the global optimal solution quickly.

# Conclusions

When more cores are integrated onto a single chip, the tasks can run in parallel to achieve better performance and cut down the power consumption. However, how to improve the scheduling efficiency is emerging as a new challenge. In this paper, a novel PSO based task scheduling algorithm is proposed for this problem. Optimized particle swarm algorithm is used to find the optimal scheduling solution based the system model. The experimental results show that this algorithm can improve the efficiency of the task scheduling for multi-core processors.

##### References

1. M.S. Khaira, “Micro-2010: lead performance microprocessor of the year 2010-myth or reality,” Proc of Twelfth International Conference on VLSI Design (VLSID99), Jan. 1999, pp. 157-163.
2. K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chung, “The Case for a Single-Chip Multiprocessor,” ACM SIGOPS Operating Systems Review, vol 30, issue 5, Dec. 1996, pp. 2-11.
3. B.A. Nayfeh, K. Olukotun “A single-chip multiprocessor,” IEEE Computer, vol 30, issue 9, Sep. 1997, pp. 79-85.
4. W. Cesario, A. Baghdadi, L. Gauthier et al, “Component-Based Design Approach for Multicore SoCs”, Proceedings of Design Automation Conference, 2002, pp. 789-794.
5. J. Cong, B. Yuan, “Energy-efficient scheduling on heterogeneous multi-core architectures”, Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design, 2012, pp. 345-350.
6. R. Das, R. Ausavarungnirun, O. Mutlu, et al, “Application-to-core mapping policies to reduce memory interference in multi-core systems”, Proceedings of the 21st international conference on Parallel architectures and compilation techniques, 2012, pp. 455-456.
7. M. Åsberg, T. Nolte, S. Kato, “Towards partitioned hierarchical real-time scheduling on multi-core processors”, SIGBED Rev. 11, 2, pp. 13-18.
8. J. Lee, G. Lim, S. Suh, “Preemptibility-aware responsive multi-core scheduling”, Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 748-749.
9. A. Fedorova, “Operating System Scheduling for Chip Multithreaded Processors,” PhD thesis, Harvard University, November 7, 2006
10. Y.C. Zhuang, Ce-Kuen Shieh, Tyng-Yue Liang and Jun-Qi Lee, “A Group-Based Load Balance Scheme for Software Distributed Shared Memory Systems,” The Journal of Supercomputing, 28:295–309, 2004.
11. G. Attiya and Y. Hamam, “Two Phase Algorithm for Load Balancing in Heterogeneous Distributed Systems,” In Proceedings of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pages 434–439, 2004.
12. J. Aas, “Understanding the Linux 2.6.8.1 CPU scheduler,” http:// josh.trancesoftware.com/linux/linux\_cpu \_scheduler.pdf
13. W. H. Ho and T. M. Pinkston. “A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns”, Proc. of the 9th Intl. Symp. on High-Performance Computer Architecture, p.377, February 08-12, 2003.
14. K. Srinivasan and K.S. Chatha. “ISIS: a genetic algorithm based technique for custom on-chip interconnection network synthesis”, Proc. of 18th Intl. Conf. on VLSI Design, pp. 623-628, 2005.
15. S. Murali, L. Benini and G. de Micheli. “Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees”. Proc. of Conf. on 2005 Asia and South Pacific Design Automation Conf., pp.27-32, 2005.
16. G. Ascia, V. Catania, M. Palesi. “Multi-objective mapping for mesh-based NoC architectures”. Proc. of 2004 Intl. Conf. Hardware/Software Codesign and System Synthesis, pp. 182-187, 2004.
17. D. Barcelos, E. W. Brião, F. R. Wagner. “A hybrid memory organization to enhance task migration and dynamic task allocation in NoC-based MPSoCs”, Proc. of the 20th annual conference on Integrated circuits and systems design, pp.282-287, 2007.
18. C. Chou, R. Marculescu. “User-aware dynamic task allocation in networks-on-chip”, Proc. of the Conf. on Design, automation and test in Europe, pp.1232-1237, 2008.
19. J. Hu and R. Marculescu, “Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints”, Proc. of the Conf. on Design, automation and test in Europe - Volume 1, pp.10234, 2004.
20. E. W. Briao, D. Barcelos, F. Wronski, F.R. Wagner,“Impact of Task Migration in NoC-based MPSoCs for Soft Real-time Applications”, Very Large Scale Integration, 2007. VLSI - SoC 2007, pp.296-299, 2007.
21. L. Schor, I. Bacivarov and D. Rai et al. “Scenario-based design flow for mapping streaming applications onto on-chip many-core systems”, Proc. 2012 Intl. Conf. Compilers, architectures and synthesis for embedded systems (CASES '12). ACM, New York, NY, USA, pp. 71-80.
22. Y. Wang, H. Liu and Z.W. Qin et al. “Overhead-aware energy optimization for real-time streaming applications on multiprocessor System-on-Chip”, ACM Trans. Des. Autom. Electron. Syst. Vol. 16, issue 2, Article 14, 32 pages, April 2011.
23. D.S. Zhang, D.K. Guo and F.Y. Chen et al. “TL-plane-based multi-core energy-efficient real-time scheduling algorithm for sporadic tasks”, ACM Trans. Archit. Code Optim. Vol. 8, issue 4, Article 47, 20 pages, January 2012.
24. C. Liu, J. Li and J. Rubio et al. “Power-efficient time-sensitive mapping in heterogeneous systems”, Proc. the 21st Intl. Conf. on Parallel architectures and compilation techniques (PACT '12), ACM, New York, NY, USA, pp. 23-32.
25. C. Fu, Y. Zhao, M. Li, C. J. Xue, “Maximizing common idle time on multi-core processors with shared memory”, Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE '15). EDA Consortium, San Jose, CA, USA, pp. 900-903.
26. G. Dhiman, V. Kontorinis, D. Tullsen et al, Dynamic workload characterization for power efficient scheduling on CMP systems. In Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design (ISLPED '10). ACM, New York, NY, USA, pp. 437-442.
27. K. Ma, X. Li and M. Chen et al. “Scalable power control for many-core architectures running multi-threaded applications”. SIGARCH Comput. Archit. News, Vol. 39, issue 3, pp. 449-460, June 2011.
28. S. Chen, J. Montgomery, “A simple strategy to maintain diversity and reduce crowding in particle swarm optimization”, Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO '11), Natalio Krasnogor (Ed.). ACM, New York, NY, USA, pp. 811-812.