Tailoring Genetic Algorithm for Resource Scheduling in Many-Core Processors

Xiande Hu[1,a], Jingming Li [1,b], Jiaxing Cheng[1,c]

[1]School of Information Engineering, AnHui XinHua University, Hefei 230088, China

[a]email: xhhxd2000@163.com ,[b]email: homer998@163.com , [c]email cjx@ahu.edu.cn

Keywords: Many-core processors, Resource scheduling, Genetic algorithm

Abstract. With the development of multi-core and many-core processors, how to leverage the hardware improvement to boost application performance remains a challenge for ensuring the continuously improvement of computing and data processing capabilities. Specifically, in this paper, we focus on how to efficiently assign many-core resources to computing tasks to improve the processing capability of systems. We propose to employ Genetic Algorithm (GA) for many-core resource scheduling. Considering the issues with GA, such as the low convergence speed, low efficiency of the search process, and limitations of the genetic operators, we propose two modifications. First, to increase the discrimination of fitness value and therefore accelerate the convergence speed, we introduce the idea from Simulated Annealing (SA) to design the fitness function. Second, to deal with the limitations of genetic operators, we introduce Cellular Automata (CA), and therefore to overcome the problem of premature convergence. With the extensive experiments, our modified algorithm exhibits fast convergence and efficient performance for many-core scheduling.

Introduction

Recent years, with the development of single-core processors integration and improvement, issues emerge such as manufacturing cost, power consumption and heat dissipation. Therefore, multi-core and multi-threading technologies have been the trend of processor system [1]. Currently, there are products which integrate tens of cores on a single processor. Moreover, with the increasing development of technology and demands, processors would have hundreds and thousands of cores, which are called many-core processors [2]. The increasing number of cores in many-core processors ensures the continuously improvement of computing and data processing capabilities. However, how to leverage the hardware improvement to boost application performance is one of the tough challenges for many-core era.

Specifically, for computing intensive application tasks, it remains tricky problems how to efficiently assign many-core resources to computing tasks, in order to improve the processing capability of systems. Currently the solutions are mostly based on virtual machines [3-5]. That is, with the perception of the underlying topology, manage resources with a running queue of CPUs and create a mapping form virtual CPUs to physical CPUs. However, as indicated by [6], as the number of cores increases, the resource utilization of the runtime system is reduced, and thus the scalability of the system is restricted since the performance of application cannot increases with the growth of cores.

To this end, in this paper, we to use Genetic Algorithm (GA) [7] for scheduling resources in many-core systems. Indeed, GA has been widely employed in scheduling problems [8-11]. However, there still exist some limitations. For example, the convergence speed is slow, the efficiency for later stage of the search process is low, and the limitations of the genetic operators restrict the proceedings of the algorithm as well [12].

Therefore, we propose some modifications for GA. First, to increase the discrimination of fitness value and therefore accelerate the convergence speed, we introduce the idea from simulated annealing (SA) [13] to design the fitness function. Second, to deal with the limitations of genetic operators, we introduce cellular automata (CA) [14], and therefore to overcome the problem of

premature convergence.

Problem Statement

We consider the resource scheduling with multiple applications. Suppose $T_{app} = \{T_1, T_2, ..., T_n\}$ denotes the set of application tasks, where n is the number of tasks, and $R = \{R_1, R_2, ...R_m\}$ denotes the set of resources (i.e., cores), where m is the number of cores. Let $R_{\min i}$ denote the minimum number of cores for task T_i.

Suppose t_{ij} denotes the execution time of task i on resource j, and x_{ij} denotes task i is assigned to resource j, that is:

$$x_{ij} = \begin{cases} 1, & \text{if task } i \text{ is assigned to resource } j; \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

$$t_j = \sum_{i=1}^{n} x_{ij} t_{ij} \,. \quad (2)$$

Therefore the objective of many-core resource scheduling is to minimize the overall makespan of all cores:

$$\min \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} t_{ij} \,, \quad (3)$$

where m, n are the number of cores and tasks respectively, and

$$\sum_{j=1}^{m} t_{ij} \geq R_{\min i} \,. \quad (4)$$

Resource Scheduling Algorithm Based On GA

Chromosome coding.

As discussed in [15], typical task-resource coding strategy is: the length of chromosome denotes the number of tasks, and the value of every gene corresponds to the resource for each task. As shown in Figure 1, where T_i denotes the i-th task, and P_i denotes the resource that is assigned to T_i. However, in many-core systems, the number of resources m and tasks n are typically very big. To avoid the rapidly growth of the length of chromosome when the tasks increase, we use a real-coded strategy instead. As shown in Figure 2, where R_j denotes the j-th resource, X_{ij} denotes the list of tasks assigned on R_j, and between the task assignments of resources there is a 0 as a delimiter. For example, given a coding:

$$\{2,3,0,1,5,6,0,4,7,8,9,0,10\} \,. \quad (5)$$

That means, there are 4 resources and 10 tasks in the system. Tasks 2 and 3 are assigned to resource 1, tasks 1, 5 and 6 are assigned to resource 2, tasks 4, 7, 8, and 9 are assigned to resource 3, and task 10 is assigned to resource 4.

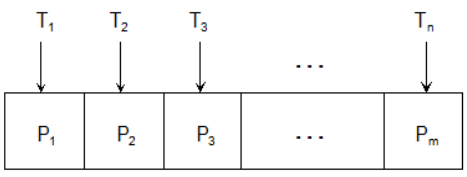

Fig.1. Typical task-resource coding

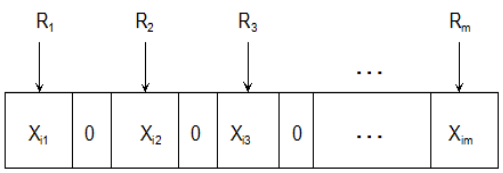

Fig.2. Our task-resource coding

Initial population.

We use a greedy strategy to generate the initial population. Specifically, sort all tasks by the priorities, and assign idle cores to the task with the highest priority first, and then the next priority, until all the tasks are assigned or no more idle cores are available.

Fitness function.

If directly use objective function as the fitness value, that is:

$$f = \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} t_{ij} \quad , \tag{6}$$

The degree of distinction between different objective function values would be small. Accordingly, the convergence speed of the algorithm would be very slow, and the execution time is therefore increasing.

To solve above problem, we introduce simulated annealing (SA) for scaling, and define the fitness function as follows:

$$f = e^{value \cdot (-a)} \cdot 10^{\alpha} \quad , \tag{7}$$

Where $value$ is the value of Equation (5), α is the adjustment for fitness range, and a is the discrimination factor of fitness value. The smaller a is, the more discriminative the fitness is. The function monotonous of fitness function and objective function is opposite. That is, when the objective function gets a minimum value, the fitness function would get a maximum. In this way, we can guarantee the makespan of the optimal solution found by our GA algorithm is minimized as in Equation (3).

Selection and crossover operators.

In order to overcome the problem of premature convergence, we introduce CA to improve the operators, which ensure the populations evolve toward the desired goal.

Selection operator preferentially selects outstanding individuals to be the next parent population for further crossover and mutation operations. A common way to define selection operator is to select the individuals with the best fitness values. Another method is roulette selection method. Suppose the fitness value of individual i is $f(i)$, and the corresponding chip for roulette is $c(i)$. Therefore, the probability of i to be selected is $p(i) = c(i) / \sum_{i} c(i)$.

Typically, crossover operator is responsible for portfolio optimization, mutation operator performs an extensive search over the entire space, and selection operator return best fitness value solutions. However, only crossover operator can generate new values. Moreover, the crossover in the whole solution space is difficult for accurate local search. Therefore, the solutions are usually lack of precision.

To this end, we introduce CA for selection and crossover operators. Define the state of cells as one of the solutions and corresponding objective function values. And the rules determine the next state of cell based on its current state and neighborhood cells. Select the cell with maximum fitness values as the value of next state, and the crossover operator is implemented as the exchange between the cell and its eight neighbors. The flow chart of modified GA algorithm with CA is shown in Figure 3.

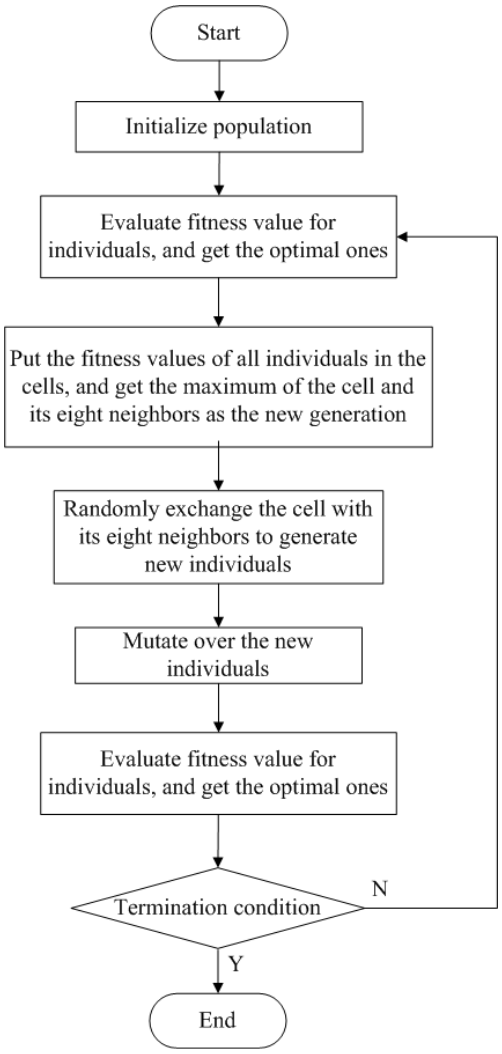

Fig.3. Flow chart of modified GA

Experiment

In order to evaluate our many-core scheduling algorithm, our experimental setup is as follows. We use NVIDIA Quadro 4000GPU with 256 cores, and the operating systems is Linux. We test resource scheduling with (1) matrix addition, (2) matrix factorization and (3) Fourier transformation tasks, and the scales of problems are shown in Table 1.

In this experiment, we have four comparison methods: (1) GA scheduling algorithm; (2) SA scheduling algorithm; (3) GSA scheduling algorithm [16]; and (4) our proposed modified algorithm. We use MATLAB to implement all algorithms.

Table1. Description of test tasks

Name	Scale (n)	Description
MA	1024	matrix addition
MF	4096	matrix factorization
FT	10240	Fourier transformation

Figure 4 shows the execution time of three tasks for four methods. Besides, Figure 5 shows the scheduling efficiency of three tasks, where the efficiency is calculated as:

$$\eta_i = \frac{\sum_{j=1}^{m} x_{ij} q_{ij}}{\sum_{j=1}^{m} x_{ij} t_{ij}}, \quad (8)$$

Where q_{ij} is the waiting time of scheduling resource j to task i.

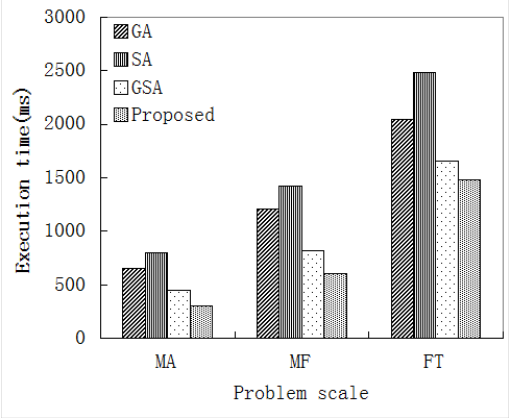

Fig 4. Execution time of three tasks

From Figures 4 and 5, we can have the following observations. First, the more complicated the task is, the more time consuming the execution is. For example, the order of execution time for three tasks is $MA < MF < FT$. Second, our proposed method outperforms other baselines, and the order of excellence of four methods is $SA < GA < GSA <$ Proposed. Third, when the computation increases, the efficiency of scheduling methods is improved. The reason is that the more computing the task needs, the more cores it requires, which accelerates the execution.

We also evaluate the execution with the increase of the number of cores. Figure 6 shows the average speedup of tasks when the number of cores increases. We can see that when more cores are involved, the speedup of our proposed method increases in a near linear way, which means that our algorithm has a good scalability in many-core systems.

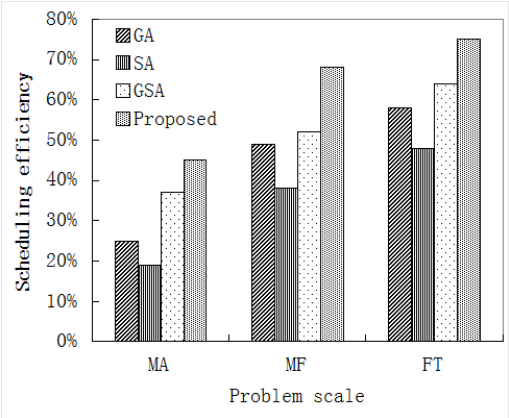

Fig 5. Scheduling efficiency of three tasks

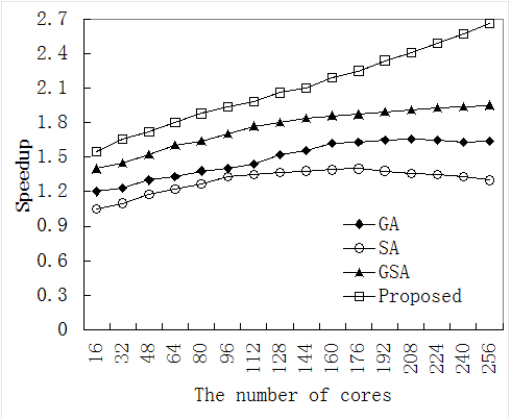

Fig.6. Speedup of four methods with increase of the number of cores

Besides, Figure 7 shows the utilization rate of cores. The utilization rate of our modified algorithm is basically stabilize around 95%, while for other algorithms, the utilization rate decreases when the number of cores increases. Therefore, our method can achieve higher rate of resource utilization.

Also, we compare the convergence of four methods in Figure 8. We can see that our proposed method convergences after approximately 100 iterations, and the convergence fitness value is the minimum of the four. Hence, we claim our modified algorithm runs faster than other baselines and achieves the best minimum objective.

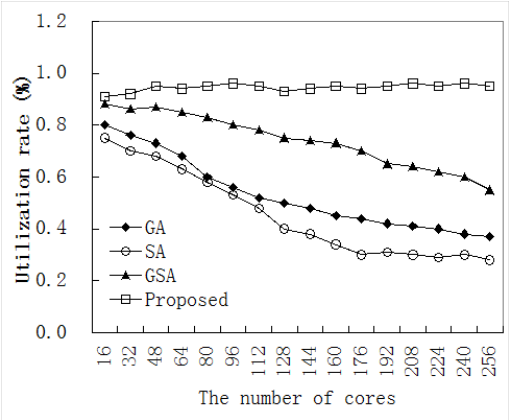

Fig.7. Utilization rate of four methods with increase of the number of cores

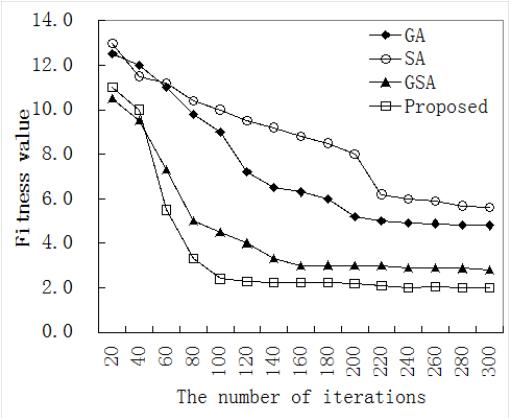

Fig.8. Convergence curve of four methods

Conclusion

In this paper, we tailor the GA algorithm for resource scheduling in many-core systems. Specifically, the modifications include: (1) in order to increase the discrimination of fitness value and therefore accelerate the convergence speed, we introduce the idea from SA to design the fitness function; (2) to deal with the limitations of genetic operators, we introduce CA, and therefore to

overcome the problem of premature convergence. Our extensive experiments evaluate the efficiency of our algorithm.

However, we don’t consider the many-core utilization across computers in clusters. Therefore, in future works, we would like to extend our research to the distributed resource scheduling with modified GA algorithm.

References

[1] Geer, David. "Chip makers turn to multicore processors." Computer 38.5 (2005): 11-13.

[2] Woo, Dong Hyuk, and Hsien-Hsin S. Lee. "Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era." IEEE computer 41.12 (2008): 24-31.

[3] Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS Operating Systems Review 37.5 (2003): 164-177.

[4] Jian, Li Yunfa Xu Xianghua Wan. "Load migration-based resource scheduling mechanism in virtual machine [J]." Journal of Huazhong University of Science and Technology (Nature Science Edition) 9 (2009): 011.

[5] Hu, Jinhua, et al. "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment." Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on. IEEE, 2010.

[6] Bhattacharjee, Abhishek, Gilberto Contreras, and Margaret Martonosi. "Parallelization libraries: Characterizing and reducing overheads." ACM Transactions on Architecture and Code Optimization (TACO) 8.1 (2011): 5.

[7] Holland, John H. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975.

[8] Hou, Edwin SH, Nirwan Ansari, and Hong Ren. "A genetic algorithm for multiprocessor scheduling." Parallel and Distributed Systems, IEEE Transactions on 5.2 (1994): 113-120.

[9] Pezzella, F., G. Morganti, and G. Ciaschetti. "A genetic algorithm for the flexible job-shop scheduling problem." Computers & Operations Research 35.10 (2008): 3202-3212.

[10] Hartmann, Sönke. "A self-adapting genetic algorithm for project scheduling under resource constraints." Naval Research Logistics (NRL) 49.5 (2002): 433-448.

[11] Gonçalves, José Fernando, Jorge JM Mendes, and Mauricio GC Resende. "A genetic algorithm for the resource constrained multi-project scheduling problem." European Journal of Operational Research 189.3 (2008): 1171-1190.

[12] Sivanandam, S. N., and S. N. Deepa. Genetic Algorithm Optimization Problems. Springer Berlin Heidelberg, 2008.

[13] Brooks, S. P., and B. J. T. Morgan. "Optimization using simulated annealing."The Statistician (1995): 241-257.

[14] Wolfram, Stephen. Theory and applications of cellular automata. World scientific, 1986.

[15] LIN Jian-Ning and WU Hui-zhong. Scheduling in Grid Computing Environment Based on Genetic Algorithm. JOURNAL OF COMPUTER RESEARCH AND DEVELOPMENT, 2004, 41(12). (in Chinese)

[16] Gaafar*, L. K., and S. A. Masoud. "Genetic algorithms and simulated annealing for scheduling in agile manufacturing." International journal of production research 43.14 (2005): 3069-3085.