Genetic algorithm for Scheduling of Data-Parallel Tasks

Yang Liu, Hiroyuki Tomiyama

Department of Electronic and Computer Engineering Ritsumeikan University

1-1-1 Noji-Higashi, Kusatsu, Shiga 525-8577, Japan

**Abstract — This paper studies a task scheduling problem which schedules a set of data-parallel tasks on multiple cores. Unlike most of previous literature where each task is assumed to run on a single core, this work allows individual tasks to run on multiple cores in a data-parallel fashion. Since the scheduling problem is NP-hard, that is, to obtain the optimal solutions require search of the overall solution space, which is very time consuming. Therefore, heuristic methods are reasonable choose to obtain suboptimal results. In this paper, an efficient method based on genetic algorithm is proposed to solve this problem. Different with traditional Genetic algorithms for task scheduling, we proposed a new representation for the chromosome of tasks scheduling and corresponding genetic operators, aim to reduce the search space and improve the computing speed. In addition to normal implementation, we also implemented our algorithm with OpenMP, show how to speeded up our algorithm further.**



(a) A task graph



(b) An optimal schedule

Figure 1. A scheduling example

**Keywords — task scheduling; multicore; data parallelism; branch-and-bound Genetic algorithm OpenMP**

# Introduction

The problem of task scheduling can be simply described as scheduling a set of tasks onto a multiprocessor system, find the minimum scheduling length under the given constraint conditions. Due to the wide deployment of multicore architecture not only in general-purpose processors but also in embedded processors, this problem has now become a more important problem than ever.

In general, task scheduling is considered a NP-hard problem. The effect of finding exact results is proved very complex, and consuming a large amount of memory and computing resources. Therefore, many heuristic approaches for task scheduling have been proposed (example). Recently, Genetic Algorithms have been widely studied as useful heuristics for obtaining high quality solutions for task scheduling problem (example).

Unfortunately, Majority of the works deal task scheduling with genetic algorithms only considered about task parallelism. Many studies has shown that, for a large class of large computational applications, exploiting both task and data parallelism yields better speedups compared to either pure task parallelism or pure data parallelism.

This paper presents an approach for task scheduling based on genetic algorithm, to solve the scheduling problem with task and data parallelism. Not only with different problem define, we also proposed a new representation for the chromosome of this problem. Our chromosome only encode information about the ordering of task execution, ignore those tasks are mapped on which cores. It is reduces greatly the size of search space and improves the performance of algorithm. The efficient genetic operators (select, crossover, mutation) corresponding to this chromosome also are presented. In additional, we implemented our algorithm with OpenMP; show how to speed-up our algorithm further.

The contributions of this paper are as follows:

* This paper presents an approach for task scheduling based on genetic algorithm, to solve the scheduling problem with task and data parallelism
* This paper proposed a new chromosome representation to scheduling problem and corresponding crossover and mutation strategies are used to minimize the makespan.
* The implementation with OpenMP for accelerating our proposed algorithm is discussed.

This paper is organized as follows. Section II formally describes a scheduling problem addressed in this paper, and Section.

# Problem Definition

Table 1. Basic terms of genetic algorithm

|  |  |
| --- | --- |
| Terms | Meaning |
| *Environment* | Problem |
| *Individual* | Solution to a problem |
| *Chromosome* | Representation for a solution |
| *Population* | Set of solutions represented by chromosome |
| *Gene* | The basic element for a chromosome |
| *Fitness* | The degree of adaptation for individual to his environment |
| *Selection* | The operation of choosing the to be parents |
| *Crossover* | The operation of producing child |
| *Mutation* | The operation of randomly alter genes |

This section defines a task scheduling problem addressed in this paper.

## Problem Description

This work assumes homogeneous multicore processors. An application is modeled as an acyclic directed graph (DAG), so called a task graph, where a node represents a task and a directed edge represents a flow dependency between two tasks. Figure 1 (a) shows an example of a task graph. In this graph, tasks labeled “*S*” and “*E*” are dummy tasks which do not perform any meaningful computation. Tasks *S* and *E* denote an entry point and an exit point of the application, respectively. Two integer values are associated with each task. The first number denotes the degree of data parallelism of the task. In other words, the number denotes the number of cores which are necessary to run the task. We assume that the degree of data parallelism is decided by programmers, and how to decide it is out of scope of this paper. The latter number on each node denotes the execution time of the task. For example, task 1 runs on 3 cores, and it takes 30 time units to complete the task.

Given a task graph, task scheduling decides when and on which core each task is executed in such a way that the overall schedule length is minimized, while meeting constraints on flow dependency among tasks and the number of available cores. Figure 1 (b) shows one of optimal schedules on four cores for the task graph in Figure 1 (a).

# GENETIC ALGORITHM

Genetic algorithm was first invented by Holland [ref], it is a meta-heuristic inspired by the processes observed in natural selection. This algorithm think of set of candidate solutions for a problem as biological population, the fitness of each individual is evaluated, according to Darwin's theory: "Survival of the fittest". The fitter ones are more likely selected and produce next generations. During this breeding process, the spontaneous mutations occur, may create individuals better adapt the environment.

The basic terms of genetic algorithm used in this paper are shown and defined in Table 1.

The basic procedure of the genetic algorithm is as follow:

1. *Initialization*: Generate the initial population
2. *Calculation of the fitness*: The fitness of each individual is calculated according by the definition of problem.
3. *Selection*: Select the adapted individuals as parent for the next generation
4. *crossover*: New individual are produced.
5. *Mutation*: Alter genes for individual
6. Go step 2 until the stopping criteria reached.

# PROPOSED ALGORITHM

The basic procedure of the genetic algorithm in describe in Section III is used in our study. We will show the detail in this chapter.

## Chromosome design

In genetic algorithm, chromosome is a set of strings which representing a potential solution for a problem. Finding an adequate chromosome is one of the most important issue for a successful application of genetic algorithms. Due to all genetic operators are defined on chromosome, a good chromosome representation will make the genetic operators easier to implement, and limit the unnecessary search space. Several different types of chromosomes are used in previous works for task scheduling problem, all of them contain the task scheduling and mapping information, means that both the ordering of tasks execution and tasks are mapped on which cores are encoded in chromosome.

This kind of chromosome may not very efficient for task scheduling with task and data parallelism, because task can be mapped on multiple cores, therefore, the length of chromosome may tend to very long. We intend to find a more compressional representing of chromosome. Our proposed chromosome only encodes information about the ordering of task execution, ignores tasks are mapped on which cores, this representation also reduce greatly the size of search space and improves the performance of algorithm.

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode  
:

The proposed representation of chromosomes is an array with N elements. N represents the number of tasks. this array determines the sequence for the processing of the tasks. Figure 2 shows an example of the proposed chromosome. In Figure 2, task1 (T1) will be scheduled first, the next one is task2 (T2), and so on.

Another very important thing is the precedence relation between tasks must be kept in our chromosome. It means that in a valid chromosome tasks must be placed in before their descendants.

## Initialization

Our algorithm begins with a randomly generate a set of candidate solutions represented by chromosome which defined in (1). The detail of initialization is explained in the below:

this algorithm of initialziation guarantees that all the generated chromosomes are valid

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode

pseudocode  
:

## Fitness function

The fitness function is used to decode a chromosome and assigns it a fitness value, we use a deterministic algorithm to schedule the tasks according to chromosome and task graph. This algorithm also restores the mapping information, that is, tasks are mapped on which cores. The pseudocode is shows as below:

Beacuse the task scheduling algorithm is aim to minimize the overall scheduling length, obviously, the fitness function can be difined as follow:

Figure 2. A chromosome example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T1 | T2 | T4 | T5 | T3 |

## Selection

The selection operator is guided by the fitness value of each chromosome calculated by (3). There are chromosome with better fitness value have a larger probability to survive. Different approaches were used in the selection operators such as roulette wheel selection, rank Selection and Steady-State Selection. Our algorithm use roulette wheel.

In roulette wheel selection, each chromosome in the population is allocated a segment on a virtual roulette wheel of a size proportional to its fitness.

The adapter chromosome with a larger segment, it means this chromosome more likely to be selected when the wheel is spun.

This size of segment is calculated as below:

## Crossover

The crossover operator is analogous to the biological crossover. Two chromosomes are chosen from the population by selection described in D, the child chromosomes are produced from them.

Due to our chromosome represent the order of task execution, simple exchange part of gene between two chromosomes may produce invalid chromosomes, we use the following method to ensure the generated chromosomes is valid:

-------------------

pseudocode

-------------------

## Mutation

Mutation operator will randomly alter one or more gene values. In genetic algorithms, selection operator will remove bad chromosomes, but lose the diversity in the population. Mutation is a very important mechanism to recover it. Hence, the mutation operator gives us the possibility of producing better child than their parents. Our mutation operator also guarantees that after mutation the chromosomes are valid.

The details about mutation are given in follow:

-------------------

pseudocode

-------------------

# Experiments

The proposed algorithm was implemented in C++. We evaluated on arbitrary task graphs of 50 tasks. The task graphs we used are derived from Standard Task Graph (STG) [9]. We conducted all experiments on Intel core i7 (i7-4790K, 4.00Hz) and 32GB memory on Ubuntu 14.04.

.

In the next set of experiments, we compared our branch-and-bound algorithms with two existing heuristic ones. One is the PCS algorithm [4] and the other is the dual-mode algorithm [5]. We used 20 sets of 50 tasks and [another](file:///C:\Users\scomu\AppData\Local\Youdao\Dict\Application\7.0.1.0214\resultui\dict\result.html?keyword=another) 20 sets of 100 tasks from [9]. The number of cores was changed from two to eight. Since tasks in STG do not assume data parallelism, we randomly assigned the degree of data parallelism to the tasks.

The results of task sets with 50 tasks are shown in Table 3 and 100 tasks are shown in Table 3. In Table 2 and 3, the best solution is marked in red. The “X” means that B&B cannot find the optimal result within 12 hours. In this case, the results are surrounded by parentheses.

As the Table 2 and 3 shows, although for some task sets B&B cannot find the optimal result in a practical time due to the huge computational cost, B&B always yields the best schedule results compare with heuristic algorithms.

The runtimes of the PCS and dual-mode algorithms were always less than 1 second. On the other hand, the runtime of our branch-and-bound algorithm significantly varied depending on the task graph. For task sets with 50 tasks on 2 or 4 cores(Table 2A and 2B), the runtime of our algorithm is less than 1 minute in most cases. Nevertheless, along with the increasing numbers of tasks and core numbers, the average runtimes also increased. For task sets with 100 tasks on 16 cores(Table 3D), 50% task sets are failed to yield an optimal result in a practical time.

# Conclusions

In this paper, we proposed a branch-and-bound algorithm for a task scheduling problem which takes into account both task-parallelism and data-parallelism. We presented four rules to prune non-optimal branches. The experiments show that our algorithm could find best schedules in a practical time for most task sets with tasks under 50.

However, for task sets with more tasks, for example, with 100 tasks, our algorithm may fail to obtain optimal results in a reasonable runtime due to the huge computational cost. In future, we plan to use parallel computing to accelerate our algorithm, in order to solve tasks sets with more tasks in a shorter runtime. we also plan to formally prove the correctness of our algorithm.

# Acknowledgment

This work is in part supported by KAKENHI 15H02680.

# References

1. S. B. Hassen, H. E. Bal, and C. J. H. Jacobs, “A task and data-parallel programming language based on shared objects,” *ACM Trans. on Programming Languages and Systems.*, vol. 20, no. 6, 1998.
2. S. Ramaswamy, S. Sapatnekar, and P. Banerjee, “A framework for exploiting task and data parallelism on distributed memory multicomputers,” *IEEE Trans. on Parallel and Distributed Systems.*, vol. 8, no. 11, 1997.
3. E. G. Coffman, *Computer and Job-shop Scheduling Theory*, Wiley, 1976.
4. Y. Liu, L.Meng, I. Taniguchi, and H. Tomiyama, “Novel list scheduling strategies for task graphs with data parallelism," *International Journal on Networking and Computing*, vol. 4, no. 2, 2014.
5. Y. Liu, L. Meng, I. Taniguchi and H. Tomiyama, “A dual-mode scheduling algorithm for task graphs with data parallelism,” *Asia Pacific Conference on Circuits and Systems*, 2014.
6. H. Kasahara and S. Narita, “Practical multiprocessor scheduling algorithms for efficient parallel processing,” *IEEE Trans. on Computers*, vol. C-33, no. 11, 1984.
7. S. Fujita, “A branch-and-bound algorithm for solving the multiprocessor scheduling problem with improved lower bounding techniques,” *IEEE Trans. on Computers*, vol. 60, no. 7, 2011.
8. O. Sinnen, A. V. Kozlov, and A. Z. S. Shahul, “Optimal scheduling of task graphs on parallel systems,” *International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2008
9. http://www.kasahara.elec.waseda.ac.jp/schedule/   
   (Last accessed: June 2016)