Using Parallel Genetic Algorithm to Solve 0-1 Knapsack Problem with Pthreads

Mengxi Liao Jianwei Xu

Department of Electrical and Computer Engineering
University of Toronto
{mengxi.liao, jianwei.xu}@mail.utoronto.ca
April 21, 2013

Abstract

The Genetic Algorithm (GA) [1] is a search heuristic utilized to generate useful solutions of optimization and search problems. However, as the performance requirement in applying this algorithm in the real world is important, fastening the algorithm to complete a job is meaningful. In this project, we tried to parallelize this useful algorithm using Pthreads with three methods, namely Single Population Model, Island Model and Hybrid Model. The Single Population Model is a shared memory model which processes multiple selections in a generation. The Island Model splits the entire population to several small pieces and assigns them to several islands to execute multiple GAs in parallel. The Hybrid Method integrates the two methods and shows advantages of the both. We used the 0-1 Knapsack problem [2] as the experimental problem and a benchmark with 10000 packages to test our parallel algorithm. Results demonstrated the best speedup using the Hybrid Method when running the implementation on a four-core machine.

1. Introduction

Genetic Algorithm (GA) is a heuristic search algorithm imitating natural evolution, which is used all across sectors of engineering and science to generate useful solutions of optimization and search problems. Many problems including NP complete can be solved by GA. Applications to GA include many fields such as bioinformatics, economics, mathematics, etc.

In a GA, a population of candidate solutions is randomly generated for an optimization problem. These candidate solutions are referred as individuals. Each individual has a value representing the fitness of this solution to the problem. After initialization, the GA processes generations in an evolution, in each of which the population experiences a series of processes imitating natural evolution, including selection, crossover, and mutation. The goal of these operations is to save the individuals with ideal fitness in the population, improve the diversity of the population, and generate more optional solutions for the problem. When the GA completes enough generations of evolutions, an individual with the best fitness will be selected as the solution of the problem.

Generally, GA is able to find solutions with high quality in reasonable amount of time. However, because they are often applied to handle difficult problems with large

datasets, GA requires large amount of time to produce a reasonable solution. Consequently, many methods of enhancing its performance are created and among which the parallel implementation is one of the most promising choices.

In this project, three methods are implemented to parallelize the general GA to solve the 0-1 Knapsack problem. The first method is the Single Population Model in which all the threads will share the whole population and execute multiple selections concurrently. The second method, Island Model, splits the whole population to several pieces, and assigns them to multiple islands. Each island executes an independent GA. In the end, the best result of all the islands will be selected as the final solution. The Hybrid Model combines the two models. Outside it runs the Island Model while in each island it runs Single Population Model. Experimental result showed that the Hybrid Model has advantages of the two previous models.

We used 0-1 Knapsack Problem as the problem with a benchmark containing 10000 packages to test our three implementations of PGA. The result is encouraging. All three methods manifested different extents of speedup with multi-cores. With four cores, the best performance reached almost five times speedup with the Hybrid Model. Except from the number of threads, the number of population is another major factor affecting the performance of the algorithms.

The report is organized as follows: in section 2, background and overview implementation of GA and the previous work on Parallel GA are provided. In section 3, the design details of our methods parallelizing GA described. The experimental environment and metrics will be illustrated in section 4. The performance results of the parallelized models are analyzed in section 5. Finally, general conclusions are given in section 6.

2. Background and Previous Work

In this section, we outline the general concept and our implementation of Genetic Algorithm in section 2.1. The previous work of parallel Genetic Algorithm (PGA) is discussed in section 2.2. In section 2.3, we provide a quick introduction of 0-1 Knapsack Problem and a brief overview of Pthreads will be provided in section 2.4.

2.1 Genetic Algorithm

Genetic Algorithms are search algorithms based on the principles of the natural evolution which contains different processes, including selection and recombination. In a GA, a population of candidate solutions of a specific optimization problem will be randomly generated. In the following pages, we will call these candidates individuals. Generally, each individual is encoded as a bit string and has a value of fitness. The value of fitness represents the quality of the solution. After generating the initial population, the algorithm starts an evolution, an iterative process, with the population in each iteration named generation. In each generation, a value of fitness representing the solution quality of a specific individual is evaluated by a fitness function defined by users. Then the algorithm executes several processes in the

current generation, which aims to produce a population with higher average and best fitness for the next generation.

These processes include:

1. Selection: Selection is a method used to selecting the individuals from the current generation to the next generation. Typically an individual with higher fitness is more likely to be selected. In our implementation, we also select the best individual to the next generation. This method can prevent the best individual from being abandoned and will accelerate the speed of increasing the best fitness from generation to generation.

2. Crossover: The individuals selected from the current generation will have certain probability to participate in crossover. Crossover provides a method of combining two parent individuals to form a child individual. In our implementation, for two n-bit individuals A and B, we randomly choose a number k between 0-n, and create a new individual with the first k bits taken from A and the last n-k bits selected from B.

3. Mutation: Before putting an individual into a new generation, there is probability to process mutation to flip one or more bits of the individual.

The algorithm terminates when a certain number of generations are produced or a satisfactory fitness level has been reached for the problem.

2.2 Previous work of Parallel Genetic Algorithm

As GA is naturally parallelizable, for better performance to produce an individual with high fitness, several methods have been created to parallelize the algorithm. In the related research [3][4], Fine-grained and Global PGAs are discussed, which are similar to our implementation. The major topic of PGA is whether the population should be split to smaller population processed independently. In a solution set, the good solutions may disperse on several different points and only one point is the optimal individual. Running several populations concurrently will prevent all the individuals from converging to only one point so that it may enhance the probability to find the best solution. In our implementation, we did not notice this phenomenon probably because of our test problem and benchmark. The investigation of this phenomenon will be put into our future work.

2.3 Introduction of 0-1 Knapsack Problem

The 0-1 knapsack Problem is a problem of combinatorial optimization. The concept is that given a specific weight and a set of items each of which has a weight and a value, determine which items should be included in a collection so that the total weight of the items selected is not larger than the specific weight and the total value of all the selected item is as large as possible.

2.4 Overview of Pthreads

POSIX Threads, usually referred to Pthreads, is a POSIX standard for threads. It provides the basic C functions for the management of threads, including mutexes, condition variables and synchronization using barriers or read/write locks.

3. Implementation

In this section, the details of our implementations of PGA will be described. Several key points of our sequential implementation to solve 0-1 knapsack problem will be discussed in Section 3.1. Then the implementations of three different methods, namely the Single Population Model, the Island Model and the Hybrid Model are displayed in Section 3.2, 3.3, 3.4 sequentially.

3.1 Sequential Implementation

This section will describe several points of our implementation different from the general GA.

Firstly, in order to use GA to solve a problem, the first step is defining an encoding method for a solution candidate and a fitness function to evaluate the candidate. For 0-1 Knapsack problem, a solution candidate is a set of items selected from the whole item set. Every item in the whole item set has an index. For each candidate solution, we encode it as a binary string. The position of a bit represents the index of the item. If the item is selected, we mark the bit as 1; otherwise, we mark it as 0. In terms of the fitness function, the definition is the sum of the values of the selected items. For example, a problem contains 3 items, t1, t2 and t3. The provided weight is W. The values of them are v1, v2, v3 and the weights are w1, w2, and w3. A solution candidate is 110, which means t1 and t2 are selected but t3 is not taken. The fitness of this individual is v1+v2.

Secondly, an additional procedure of removing the dead individuals is added in the end of each generation. A dead individual represents a solution candidate whose sum of weights is larger than the provided weight. In this case, the position of the individual will be replaced by a randomly created new individual.

Thirdly, after each generation, the best individual will be selected to the next generation without participating in the other GA processes like crossover or mutation. This method is used to guarantee that the best individual will be selected to the next generation so that the best fitness will not be lower than the previous generations.

3.2 Single Population Model

The first method of PGA is a shared-memory model in which all the threads share only one population. In each iteration of GA, a certain number of individuals from the current generation are selected to the next generation. The processes of selection,

crossover and mutation for different individuals are independent. Therefore, these processes can be executed in parallel. In our code, we created a certain number of global threads and reuse them every generation. In our experiment, this method gave good speedup with increasing number of threads.

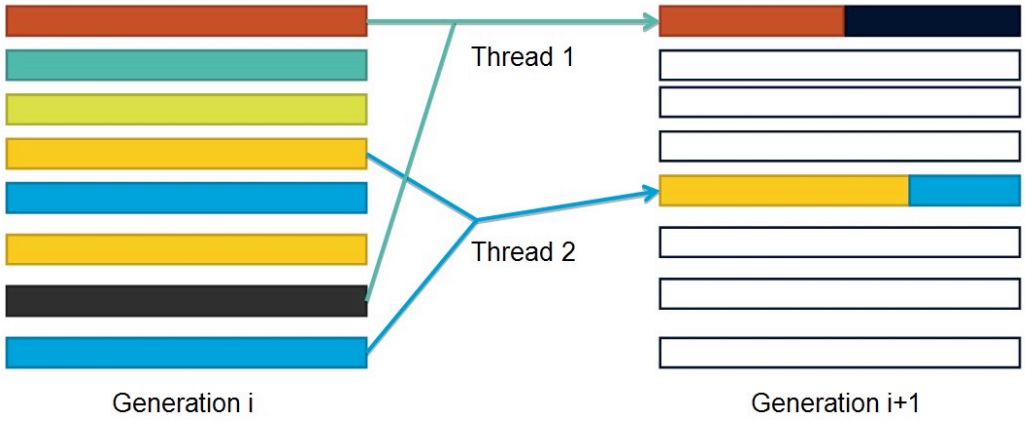

Figure 1: *Example of Single Population Model*

3.3 Island Model

In the Island Model, the basic idea is splitting the entire to small pieces and assigning them to multiple islands. Each island is assigned with a thread to execute an independent genetic algorithm independently. After all the islands complete, the individual with the best fitness among all the islands is selected as the final result.

However, since the size of population in each island is much smaller than the whole population, an island may not have enough individuals to produce a good solution. The rate of convergence shown in our experiment is very low. In order to let more individuals participate in the evolution of different islands, we used migration, a method to move multiple individuals from an island to another island. These individuals are compared with the original individuals in the destination island. If the fitness of a local individual is lower than the fitness of the external one, it will be replaced by the external individual. In our implementation, we used quick sort to sort the individuals based on their fitness before migration. After that, the first certain number of individuals will be selected and migrated to another island. The reason for selecting the individuals with high fitness is to enhance the average fitness of the other island. The number of individuals to migrate is decided by the migration rate which is defined by users. The relationship between migration rate and performance will be discussed in the experiment and results section.

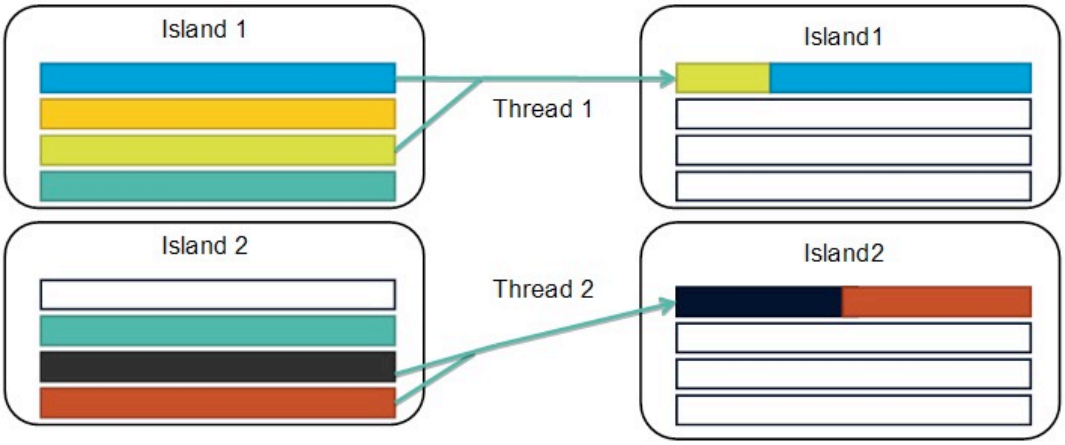

Figure 2: *Example of Island Model*

Our original design used a barrier provided by Pthreads to stop all the islands and then performs migration. This method is realized to be very wasteful as it is unnecessary to stop others when an island needs migration. Therefore, we removed all the barriers and apply unsafe migration. However, another problem was noticed that the speed of different islands become very different. For example, an island may have completed 1000 generations while another has only executed 300 generations. To solve this problem, we still use a barrier to stop the islands every certain number of generations (e.g. 150 generations) but not every one. When an island reaches the barrier, it has to wait for the others. This method can guarantee that all the islands have similar speed without interrupting each other too frequently.

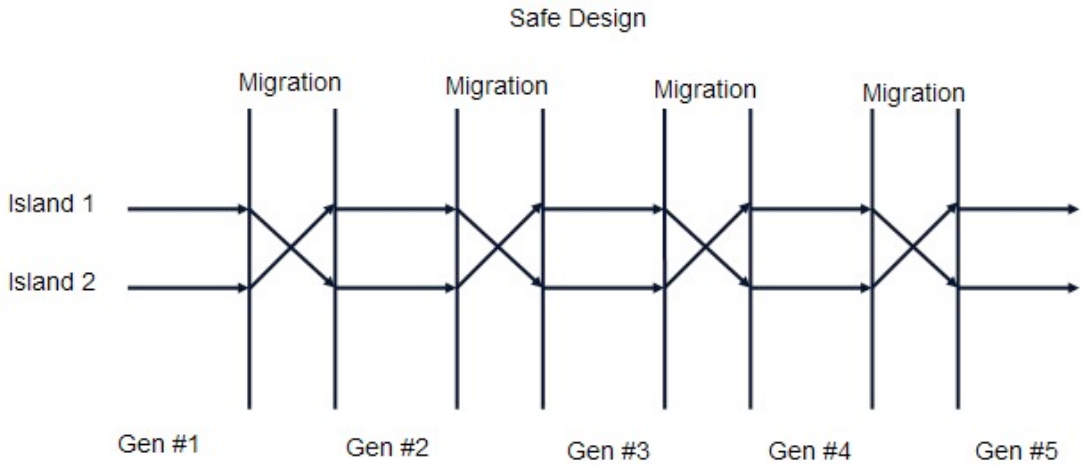

Figure 2: *Safe Migration*

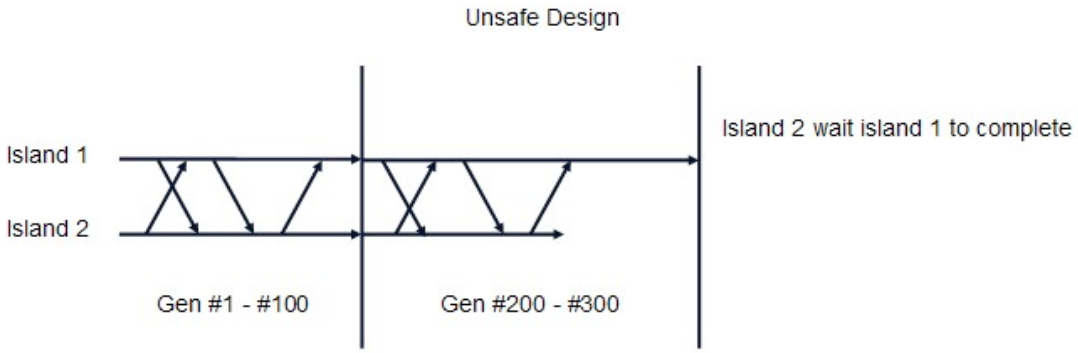

Figure 3: *Unsafe Migration*

In our experiment, this method did not show good speedup with increasing number of threads for the reasons that the migration has overhead and the more islands we use, the less individuals can the algorithm assign to an island. However, the Island Model required less generations to produce a solution with good fitness when the number of islands is 2 or 3.

3.4 Hybrid Model

The idea of Hybrid Model is direct. Outside it applies the Island Model and with each island inside it uses Single population model. In our evaluation, this method showed both advantages of Single Population Model and Island Model. Smaller number of generations was required to obtain a good solution and it also showed good speedup with increasing number of threads. As discussed before, the number of islands should be small to 2 or 3. If there are more cores available, they can be assigned to increase the parallelism in each island.

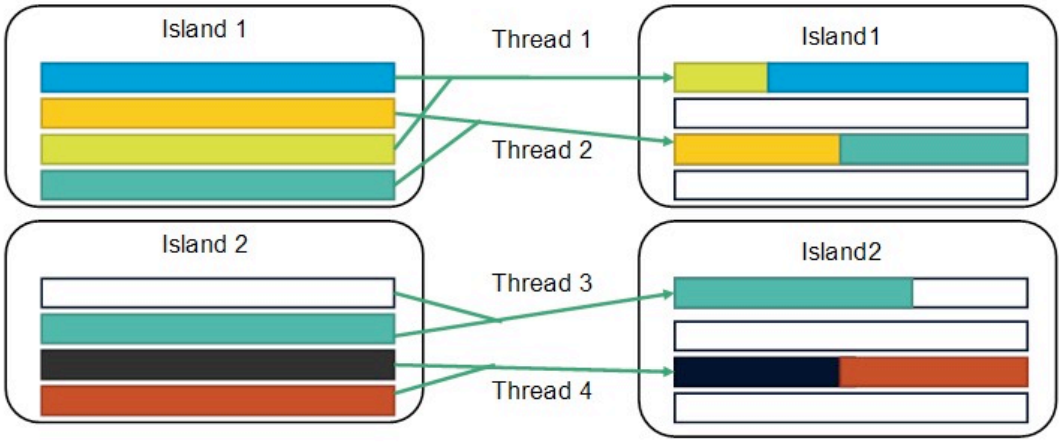

Figure 4: *Hybrid Model*

4. Experimental Environment

The experiment is conducted on UG Machines, Intel Core 2 Quad 2.66GHz and 6GB RAM, with Debian system installed. As introduced above, we measured three

models of GA which are single population model, island model and hybrid model. We used 0/1 knapsack problem as our test problem and evaluated the execution time versus different parameters such as number of threads, number of islands, size of population and migration rate.

5. Experiments and Results

In this Section, the metrics and performance of all three methods will be sequentially described in section 5.1, 5.2, and 5.3. Section 5.4 will show the final performance comparison of the three models.

5.1 Performance of Single Population Model

Due to the nature of the models, the performance of each model has its distinct features. Typically, the size of the population may play a significant role of GA. Generally, larger population leads to better convergence performance, in another word, it requires less generations to reach the best individual. Each generation, however, takes more time to compute, since it has more individuals to deal with. Therefore, it is interesting to investigate the relation between the execution time and the size of the population and find the best population size. In our experiment, we test the algorithm with 4 different sizes of population, 36, 72, 108 and 144.

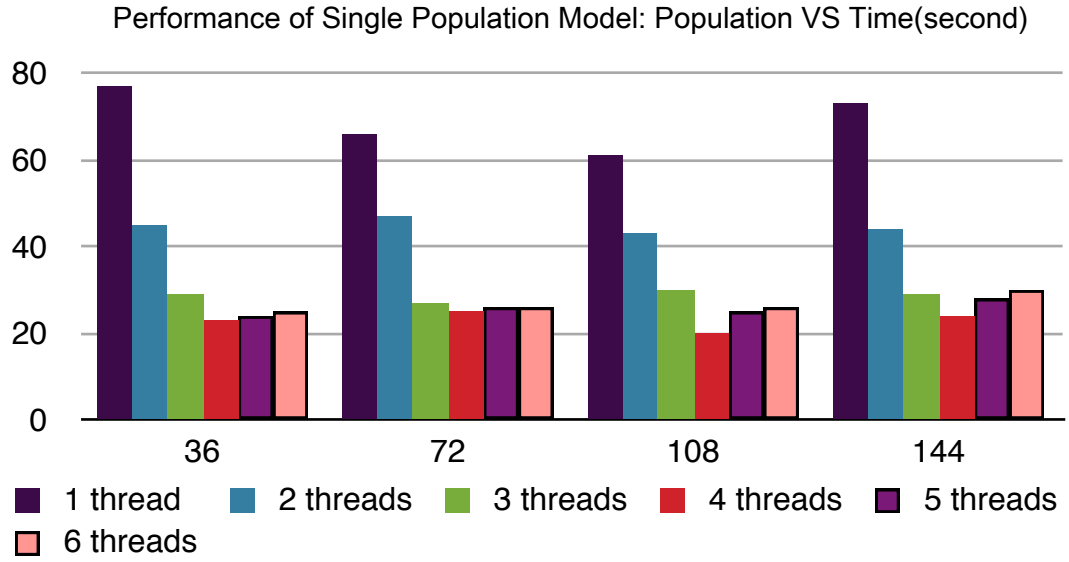

Figure 5: *Performance of Single Population Model: Population VS Time*

Figure 5 shows the performance of the Single Population Model, which has a fairly good speedup as expected. The effect of the population for the Single Population Model is also indicated in Figure 5. We observed that the speedups generally remain unaffected. It implies that with the number of population growing, the negative impact of consuming more time in each generation is neutralized by the better convergence speed.

5.2 Performance of Island Model

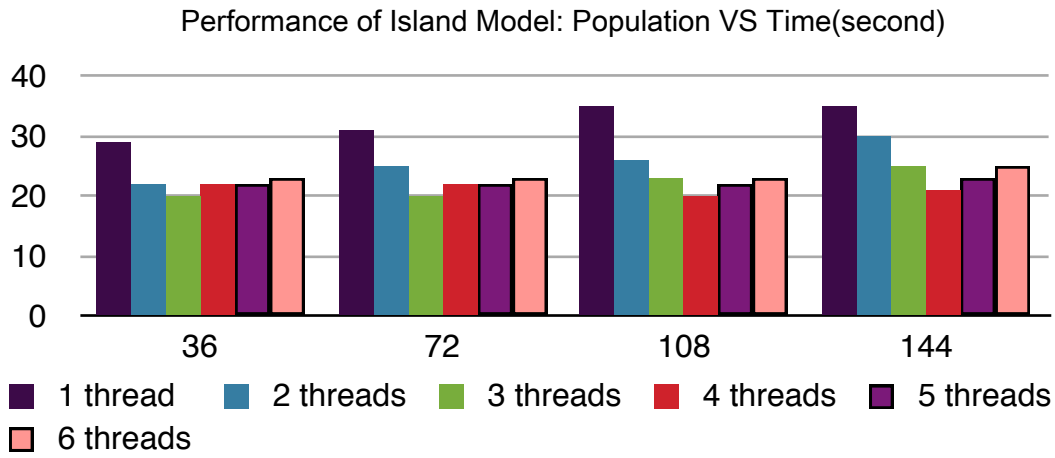

Figure 6: *Performance of Island Model: Population VS Time*

We learn from Figure 6 that the Island Model has better performance than the straightforward Single Population Model due to the migration process which substitutes bad individuals with good ones from another island. However, it is remarkable that the speedup is worse than the Single Population Model. Creating more threads only leads to small performance improvement. In addiction to that, when with low population, using more than 4 threads even brings worse performance, which is resulted from that each island does not have sufficient individuals to produce a solution with high fitness. Therefore, the computation has to iterate more generations, leading to longer execution time. Moreover, the additional process of migration adds overhead to the execution time for each generation. The overhead rises with the increasing number of islands. The experiment also illustrates the poor speedup of the model, given the fact that the more threads we create, the less individuals each island has.

Analog to the single population model, the influence of the change of the population size on performance is also not significant, expect we have some fluctuations. Thus similar to the conclusion of single population model, with larger size of population, the computation needs to iterate more generations yet each generation requires less time to compute.

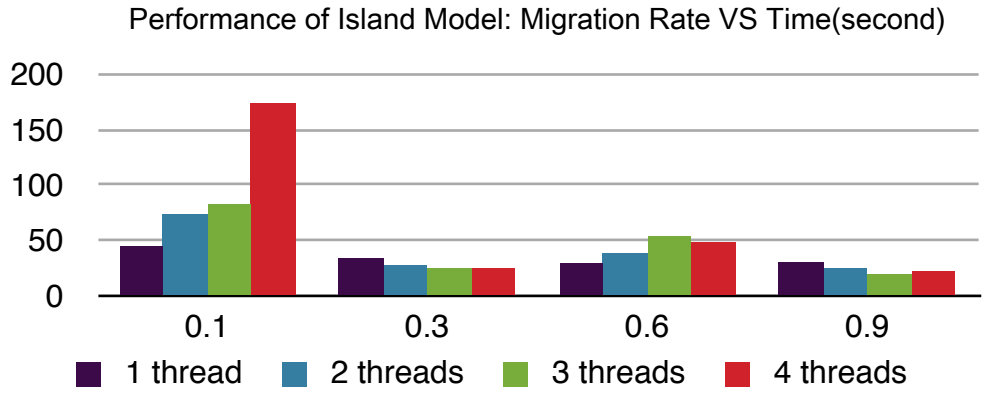

Figure 7: *Performance with different migration rate*

Besides the population size, another critical parameter is the migration rate. Migration mechanism is applied both in the Island Model and the Hybrid Model. We tested the migration rates various from 0.1 to 0.9. Figure 7 shows that migration significantly improves the performance. In general, higher migration rate is better.

5.3 Performance of Hybrid Model

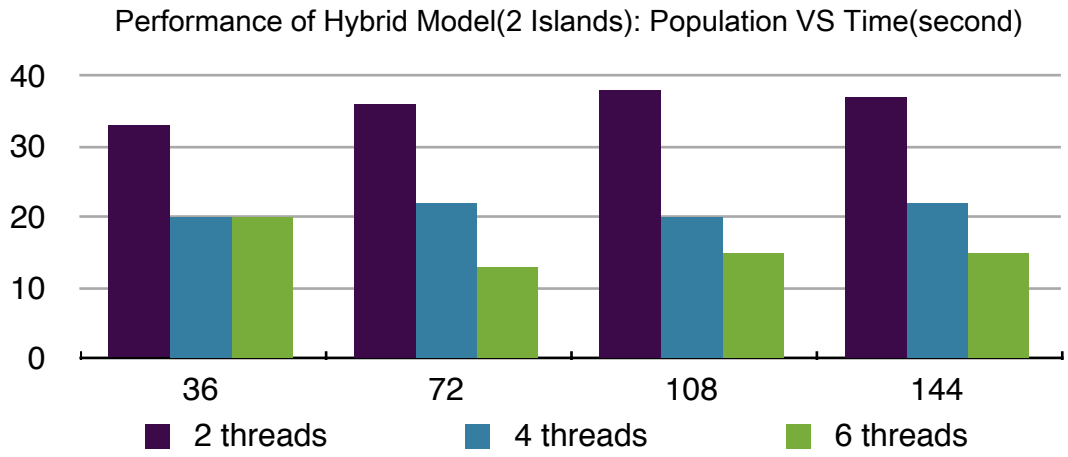

Figure 8: *Performance of Hybrid Model: Population VS Time*

The results of the hybrid model are presented in Figure 8. The Hybrid Model has better performance than the Single Population Model and the Island Model. In the experiment, we only created two islands maintained by two threads for reducing the overhead of migration. As anticipated, the model has advantages of the both models. It needs less generations to generate good solution quality and has good speedup with increasing number of threads.

Notice that the best performance occurs when we create 2 threads more than the number of cores. However, using more threads does not bring benefits for other models. The reason for this is unknown with our limit knowledge but probably because of the mechanism how Pthreads assigns threads to different cores. Similar to the previous methods, the size of population does not affect the performance significantly.

5.4 The Comparison of Performance of Different Models

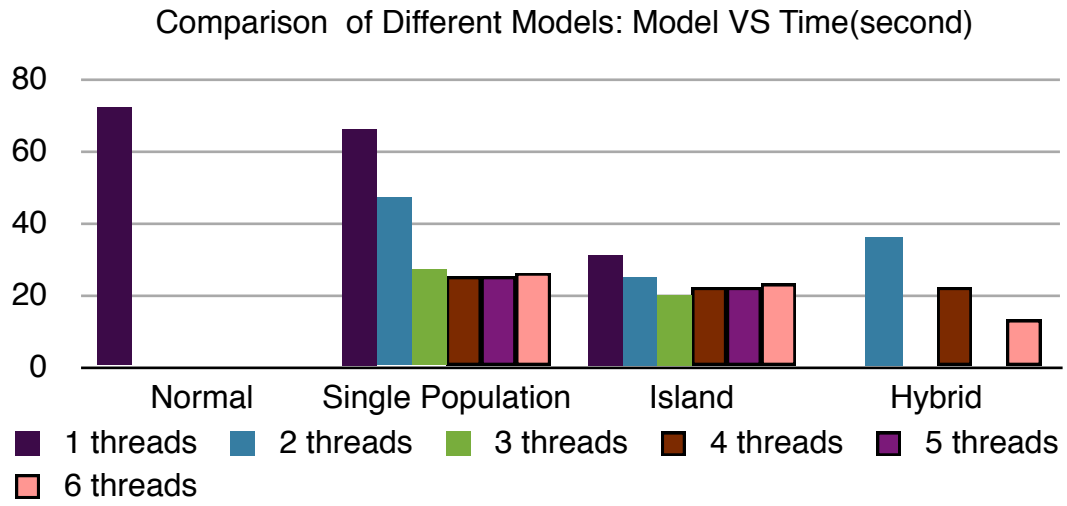

Figure 9: Comparison of Different Models

Figure 9 presents the comparison of the performance among different models. Inspired by the previous discussion, we choose the population of size 72, migration rate of 0.9 as our parameters for the Island Model and the Hybrid Model. It is obvious to indicate that the hybrid model has the best performance. It successfully reduces the execution time from 72 seconds to 13 seconds, with a quad core CPU supplied. The migration process plays an important role in the performance improvement.

6. Conclusion

Parallel Genetic Algorithm has many parameters including number of threads, number of islands, size of population and migration rate that may affect the performance for a specific problem and benchmark. This project tried to investigate the relationship of these parameters and fasten its time to solve the 0-1 problem with a large data set.

As the results demonstrated, the trade-off between the convergence rate and the time to execute each generation is a major problem in this project. Single Population Model shows great speedup while Island Model has good convergence speed. For Hybrid Model, it has advantages of the both models.

7. References

[1] John Holland, “Genetic Algorithms”, Scientific American, July 1992. Easy introduction by the father of the field.

[2] Kellerer, Hans, Pferschy, Ulrich; Pisinger, David (2004). Knapsack Problems. Springer.

[3] Erick Cantú-Paz, “A survey of Parallel Genetic Algorithms”, Department of Computer Science and Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

[4] Serial and Parallel Genetic Algorithms as Function Optimizers, V. Scott Gordon and Darrel Whiteley, Technical Report CS-93-114, September 16, 1993