Analysis of Scalable Parallel Evolutionary Algorithms

Jun He and Xin Yao

Abstract— Inherent parallelism is regarded as one of the most important advantages of evolutionary algorithms. This paper aims at making an initial study on the speedup of scalable parallel evolutionary algorithms. First the scalable parallel evolutionary algorithms are described; then the speedup of such scalable algorithms is defined based on the first hitting time; Using the new definition, the relationship between population diversity and superlinear speedup is analyzed; finally a case study demonstrates how population diversity plays a crucial role in generating the superlinear speedup.

I. INTRODUCTION

Inherent parallelism is regarded as one of the most important advantages of evolutionary algorithms. Parallelization aims at shortening the execution time of an algorithm on a parallel computing system. Different implementations of Parallel Evolutionary Algorithms (thereafter PEAs) have been proposed and studied before [1], [2], [3], [4], [5]. A review of PEAs can be referred to [3], which has made a complete survey on parallel techniques used for evolutionary algorithms.

In the study of PEAs, one of the most interesting questions is whether superlinear speedup exists and how this phenomenon happens. The superlinear speedup sometimes is controversial, however the previous studies have experienced such a speedup, e.g., [6], [4] have reported the super-linear speedup both in theory and practice.

Different from previous studies, this paper makes an initial investigation on whether and how the superlinear speedup happens in scalable PEAs. First we describe sequential, parallel and scalable parallel EAs; then discuss why the traditional speedup is not suitable to analyze the scalable PEAs, and propose a new definition of speedup for the scalable PEAs, called scalable speedup, based on the first hitting time; using the new definition, we show the superlinear speedup could happen for the wide-gap problem, where population diversity plays a key role; finally we report an experimental study to demonstrate the relationship between speedup and population size.

II. SCALABLE PARALLEL EVOLUTIONARY ALGORITHMS

This paper considers the following problem: given the binary string set B^n , a function $f : B^n \rightarrow R$ and an $f^* \leq f_{\max}$, then find an $x \in B^n$ such that:

$$f(x) \geq f^*. \quad (1)$$

Jun He is with School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. and School of Computer Science, Beijing Jiaotong University, China. (Email: j.he@cs.bham.ac.uk)

Xin Yao is with School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K. (Email: x.yao@cs.bham.ac.uk)

If $f^* = \max\{f(x)\}$, then the above problem is a maximization optimization problem. For convenience, assume that f^* is known in advance. This assumption is used to generate a simple halting criterion.

The Sequential EA (SEA) is described in the following algorithm, which uses a population consisting of N individuals. All populations form a population space $S(N)$, where N is the population size.

SEA(N)

begin

Generation counter $t := 1$;

Initialize $\xi(t)$;

while (Fitness is less than f^*) **do**

$\xi'(t) := \text{variation } \xi(t)$;

evaluate $\xi'(t)$;

$\xi'(t+1) := \text{select } (\xi(t), \xi'(t))$;

$t := t + 1$;

od

Output the result.

end

The above EA can be implemented through various parallel strategies; as a consequence, the behavior and analysis of various PEAs is different. This paper considers a fine-grained parallelization strategy in which one individual is assigned to one processor. This kind of parallelization strategy is suitable on massively parallel machines. A more detailed discussion on this implementation can be referred to [7]. The paper assumes that each processor is connected with K neighbor nodes.

For each individual, the mutation and evaluation operators can be implemented in a single processor without involvement of other processors; it is easy to be parallelized. The crossover needs a small communication between one of the adjacent processors.

The parallelization of selection operator is a little complex, dependent on what types of selection operators are used. The simplest parallelization strategy is to run several same copies of $(1+1)$ EA independently on a parallel computing system, which no communication cost is needed. Next let's investigate other selection operators. Tournament selection is one of common selection operators used in EAs. It runs a "tournament" among a few individuals, which are chosen at random from the population, and selects the winner (the individual with the best fitness). The operator compares the fitness of an individual with k individuals in its neighbor (k is called the tournament size), and choose the best individual to replace the current individual. There are k comparisons for selecting the best individual among k individuals, hence at least $\log_2(k)$ parallel steps is needed to implement the k

fitness comparisons.

Winner-take-all strategy is another selection strategy, which chooses the best individual from the whole population and bring s it in the next generation. The procedure of selection can be implemented in a parallel way but more complex than tournament selection. There are N comparisons for selecting the best individual in a population consisting of N individuals, therefore at least $\log_2(N)$ parallel steps is needed to implement the N comparisons.

Now we present a traditional “divide-and-conquer” parallelism for the sequential SEA(N), which are based on an ideal but simplified environment: we have adequate processors. Each individual is assigned to a processor, if the number of individuals is less than that of processors. The PEA(N) using N individuals is given as follows:

PEA(N)

begin

Generation counter $t := 1$;

Initialize $[\xi(t)]$ on N processors;

while (Fitness is less than f^*) **do**

$\xi'(t) :=$ variation $\xi(t)$ on N processors;

evaluate $\xi'(t)$ on N processors;

$\xi'(t+1) :=$ select $(\xi'(t), \xi(t))$ on N processors;

$t := t + 1$;

od

Output result.

end

Now we define Scalable Parallel Evolutionary Algorithms where N is the population size. Its structure is as the same as the PEA(N). But its idea is completely different from the above classical “divide-and-conquer” parallelism: if we have more processors, we will employ more individuals to work. For example, if we have N processors, then we use an EA with N individuals. This means we run the $(1+1)$ EA on uniprocessor; we use the PEA(N) on multiprocessor. In other words, the EA running on multiprocessor is different from that on uniprocessor.

Hence we will not compare the scalable PEA(N) with the sequential rival SEA(N); instead, we should compare it with the PEA(1), described as:

begin

Set generation counter $t := 1$;

Initialize an individual $\xi(t)$;

while (Fitness is less than f^*) **do**

$\xi'(t) :=$ variation $\xi(t)$;

evaluate $\xi'(t)$;

$\xi'(t+1) :=$ best from $(\xi(t), \xi'(t))$;

$t := t + 1$;

od

Output the result.

end

III. FIRST HITTING TIMES AND DEFINITION OF SPEEDUP

The speedup must be defined carefully in PEAs, as pointed by [3]. Obviously it is an average time, rather than a result

from a single run. It is also dependent on the probability distribution π of initial population. Let $T_1(\pi_1)$ be the mean execution time of sequential algorithm SEA(N) when the initial individual satisfies the probability distribution π_1 , and $T_1(\pi_2)$ the mean execution time of parallel algorithm PEA(N) when the initial individual satisfies the probability distribution π_N , then the traditional speedup can be defined by the ratio between the mean execution time $T_1(\pi_1)$ for PEA(N) on a uniprocessor and the mean execution time $T_N(\pi_N)$ of the PEA(N) on N processor.

$$s_N = \frac{T_1(\pi_1)}{T_N(\pi_N)}. \quad (2)$$

In the following we discuss the relationship between the speedup and the first hitting time. The first hitting time is the number of generations for PEA needed to find the optimal solution. It can be defined rigorously by the first hitting time. Let $\{\xi_t; t = 1, 2, \dots\}$ be the population sequence associated with an EA. Then the stopping time

$$G(\xi_1) = \min\{t \geq 1; f(\xi_t) \geq f^* \mid \xi_1\},$$

is defined as the first hitting time to find a solution when starting from an initial population ξ_1 , G is a random variable. If the initial population satisfies probability distribution π , then its expectation is

$$\bar{G}(\pi) = E[G(\xi_1)] = \sum G(\xi_1)\pi(\xi_1).$$

Without confusion, we denote $\bar{G}(\pi)$ by $\bar{G}$ in short. In the scalable PEAs using N processors, $\bar{G}_N$ is denoted to be its mean first hitting time.

The first hitting time is dependent on the probability distribution of the initial population. There are at two different measures: one is the worst-case analysis and another is the average-case analysis.

- 1) Worse-case time (all individuals in the initial population only take the worst point):

$$\bar{G}_w = \max\{E[G(\xi_1 = x)]; x \in S\}.$$

- 2) Average-case time (the initial population satisfies the uniform distribution):

$$\bar{G}_a = \frac{\sum_x E[G(\xi_1 = x)]}{|S|}.$$

In the following discussion, we ignore the subscripts a and w for convenience, only label them out when they are necessary.

The difference between the mean execution time T_N and the mean first hitting time $\bar{G}_N$ is that: T_N represents the execution time for PEA(N) to find the solution, $\bar{G}_N$ the mean generation for PEA(N) to find the solution.

Now we analyze the traditional speedup. Let's apply the Amdahl's law on the PEA(N) and obtain the maximum speedup.

Lemma 1: Let T_s be the average time on the sequential part of PEA program at each generation, T_c the mean time on the communication, and T_p the time in the parallel part of

the PEA program, if the communication cost can be ignored compared with other computation cost, then the speedup is:

$$s_N = \frac{1}{F + (1 - F)/N}, \quad (3)$$

where $F = T_s/(T_s + T_p)$.

Proof: On the uniprocessor, the total computation time is that

$$T_1 = E[G](T_s + T_p). \quad (4)$$

On the multiprocessor, the total computation time is that

$$T_N = E[G](T_s + T_c + T_p/N). \quad (5)$$

The maximum speedup that can be achieved by using N processors is

$$s_N = \frac{T_1}{T_N} = \frac{T_s + T_p}{T_s + T_c + T_p/N}. \quad (6)$$

If the communication cost T_c can be ignored, then let $F = T_s/(T_s + T_p)$, where F is the fraction of a calculation that is sequential (i.e. cannot benefit from parallelization), and $(1 - F)$ is the fraction that can be parallelized.

The above equation can be rewritten in the following form:

$$s_N = \frac{1}{F + (1 - F)/N}. \quad (7)$$

Let's estimate the speedup s_N for PEA(N) using a k -tournament selection. According to the conventional definition of speedup, we run the same algorithm, PEA(N) with the same population size, both on uniprocessor and multiprocessor systems.

Theorem 1: Let N be the number of processors. At each generation, denote the computation time of the variation operation in one individual to be t_v , the time of the evaluation operator t_e and the time of each comparison in the selection operator t_s . If PEA(N) uses a k -tournament selection, then the speedup is

$$s_N = \frac{Nt_v + Nt_e + Nt_s}{t_v + t_e + t_c + \log_2(k)t_s}. \quad (8)$$

Proof: For SEA(N) running on a uniprocessor, $T_v = Nt_v$, $T_e = Nt_e$. Since the EA uses a k -tournament selection, so at each generation, each individual needs k comparisons, and the computation time to implement the k -tournament selection is $T_s = Nkt_s$, and the total computation time is

$$T_1 = \bar{G}(Nt_v + Nt_e + Nkt_s). \quad (9)$$

For PEA(N) running on the multiprocessor, the variation and evaluation processes can be implemented in a parallel way; however, the k -tournament selection needs at least $\log_2(k)$ -parallel steps since the number of fitness comparisons is k . The total computation time is that

$$T_N = \bar{G}(t_v + t_e + t_c + \log_2(k)t_s). \quad (10)$$

The speedup is

$$s_N = \frac{Nt_v + Nt_e + Nt_s}{t_v + t_e + t_c + \log_2(k)t_s}. \quad (11)$$

From this theorem, we can see that the speedup is a monotonously increasing function of the population size N . From the function, if we have enough processors, then the speedup could be very large; however, this is unrealistic because the speedup in the real world always has a limitation. Even we have adequate processors, we still need at least one generation to evaluate the fitness function. The problem is caused by the traditional definition of speedup, which focuses on comparing SEA(N) and PEA(N) with the same population size N running on uniprocessor and multiprocessor. But the scalable EAs often use a small population size on uniprocessor, and a large population size on multiprocessor. This means the EAs running on multiprocessor are different from that on uniprocessor.

We can clarify this in more details. In the scalable parallel EAs, if we have only one processor available, and we would run PEA(1), rather than PEA(N), then the computation time is

$$T_1 = \bar{G}_1(t_v + t_e + t_s).$$

If we have N processors available and run PEA(N) with a k -tournament selection, then the computation time is at least

$$T_N \geq \bar{G}_N(t_v + t_e + t_c + \log_2(k)t_s).$$

Then the maximum speedup is upper-bounded by

$$s_N = \frac{T_1}{T_N} \leq \frac{\bar{G}_1}{\bar{G}_N} \leq \bar{G}_1.$$

Based on the above analysis, we believe we need a new definition of speedup for scalable EAs. Now we describe this new speedup. Assume that the mean execution time for an algorithm using N processors is T_N . The new definition of speedup, called scalable speedup, is the ratio between the mean execution time T_1 for PEA(1) on uniprocessor and the mean execution time T_N of PEA(N) on N processors, i.e.,

$$s_N = \frac{T_1}{T_N}. \quad (12)$$

The difference between the new speedup definition and the traditional one is that:

- The traditional definition is used to compare SEA(N) and PEA(N) with the same population size, but running on uniprocessor and multiprocessor;
- The new definition focuses on the scalability. The algorithm running on the uniprocessor is PEA(1) with only one individual, this means, no crossover, no population-based selection. The algorithm running on N processors is PEA(N) with N individuals, and this allows crossover and population-based selection. In other words, the algorithm running in uniprocessor is different from that on multiprocessor. Hence the population size is scalable to the number of processors.

As indicated before, the new speedup is limited by:

Corollary 1: The speed up is upper-bounded by T_1 .

For the EA using a crossover, the definition of speedup should make some change. The benchmark algorithm will not be the PEA(1), it should be PEA(2) using a crossover operator. The investigation on this topic will be left in the future.

Using the above definition, three different performances of scalable PEAs can be distinguished based on the speedup:

- 1) sub-linear speedup: $s_N < N$;
- 2) linear speedup: $s_N = N$;
- 3) super-linear speedup: $s_N > N$.

In the new definition, we can see that the speedup is mainly determined by the first hitting times.

Theorem 2: At each generation, denote the computation time of the variation operation in one individual to be t_v , the time of the evaluation operator t_e and the time of each comparison in the selection operator t_s . Assume the selection is a k -tournament selection, then

$$s_N = \frac{\bar{G}_1(t_v + t_e + t_s)}{\bar{G}_N(t_v + t_e + t_c + \log_2(k)t_s)}. \quad (13)$$

Assume the selection is winner-take-all selection, then

$$s_N = \frac{\bar{G}_1(t_v + t_e + t_s)}{\bar{G}_N(t_v + t_e + t_c + \log_2(N)t_s)}. \quad (14)$$

Proof: For the tournament selection, it is PEA(1) running on uniprocessor, then the total computation time is

$$T_1 = \bar{G}_1(t_v + t_v + t_s). \quad (15)$$

On multiprocessor, PEA(N) uses a population size N where N equals to the number of processors. As we have analyzed in the Theorem 1, we know the total computation time is that

$$T_N = \bar{G}_N(t_v + t_e + t_c + \log_2(k)t_s). \quad (16)$$

For the winner-take-all selection, it is PEA(1) running on uniprocessor, then the total computation time is

$$T_1 = \bar{G}_1(t_v + t_v + t_s). \quad (17)$$

For winner-take-all selection, PEA(N) uses a population size N where N equals to the number of processors.

The total computation time is that

$$T_N = \bar{G}_N(t_v + t_e + t_c + \log_2(N)t_s). \quad (18)$$

From the above computation times, we can obtain the speedup given in the theorem. ■

In general, the scalable speedup can be rewritten in an equivalent form,

$$s_N = \frac{T_1}{T_N} = \frac{\bar{G}_1(t_v + t_e + t_s)}{\bar{G}_N(t_v + t_e + t_c + t(t_s, N))}, \quad (19)$$

where $t(t_s, N)$ is a function of t_s and N , dependent on used selection scheme.

From the definition, we see that the term $\bar{G}_1/\bar{G}_N$ plays a key role in the scalable speedup. Now we call this term

generation-based speedup, or *g-speedup* in short, which is used to measure how many generations can be shortened in the scalable PEAs.

$$g_N = \frac{\bar{G}_1}{\bar{G}_N}. \quad (20)$$

In Eq. (19), we always have $t(t_s, N) \geq t_s$, hence we obtain the following relationship between s_N and g_N :

$$s_N \leq g_N.$$

Based on g_N , we can distinguish the behavior of scalable PEAs into three classes:

- 1) g-sub-linear speedup: $g_N < N$;
- 2) g-linear speedup: $g_N = N$;
- 3) g-superlinear speedup: $g_N > N$.

IV. POPULATION DIVERSITY AND SUPERLINEAR SPEEDUP

Several previous papers have shown that superlinear speedup exists for PEAs [6], [4]. Under the meaning of g-speedup, we will show the superlinear g-speedup can happen for the scalable PEAs. In the following, let's discuss a simple PEA(N), which runs N copies of PEA(1) independently. We aim at demonstrating how population diversity can bring benefit to superlinear speedup. Let's analyze the benchmark algorithm PEA(1) first.

Since population space $S(1)$ is a finite state space (in the case of PEA(1), $S(1)$ is equals to the individual space, so we write it by S in short), we know the values of fitness $f(x)$ are limited. Without loss of generality, we assume that there are $L+1$ values. Let's sort them according to the descending order,

$$f_0 = f_{\max} > f_1 > \dots > f_L = f_{\min}.$$

Define S_l to be

$$S_l = \{x \in S(1); f(x) = f_l\}, \quad l = 0, \dots, L.$$

Then the state space $S(1)$ can be decomposed into $L+1$ sub-spaces:

$$S = \cup_{l=0}^L S_l.$$

For the Markov chain $\{\xi_t; t = 1, 2, \dots\}$ associated with PEA(1), the transition probability $p(x, y)$ between $x \in S_l$ and $y \in S_k$ ($l, k = 0, \dots, L$), satisfies

$$p(x, y) = \begin{cases} 1, & x \in S_0, \quad y = x, \\ 0, & x \in S_0, \quad y \neq x, \\ \geq 0, & x \in S_l, \quad y \in S_k, \quad k < l, \\ \geq 0, & y \in S_l, \quad y = x \\ 0, & \text{else.} \end{cases}$$

Then we have the following result from [8], [9],

Theorem 3: For the PEA(1), its first hitting time is given by

$$\begin{aligned} \bar{G}(\xi_1 = x) &= 0, & x \in S_0, \\ \bar{G}(\xi_1 = x) &= \frac{1}{\sum_{y \in S_0} p(x, y)}, & x \in S_1, \\ \bar{G}(\xi_1 = x) &= \frac{1 + \sum_{k=1}^{l-1} \sum_{y \in S_k} p(x, y) \bar{G}(y)}{\sum_{k=0}^{l-1} \sum_{y \in S_k} p(x, y)}, & x \in S_l, l \geq 2. \end{aligned} \quad (21)$$

The optimization problems can be divided into two classes based on the mean first hitting time $\bar{G}(\xi_1 = x)$.

- **Narrow-gap class.** In this case, the difference of the mean first hitting times between any two conjunct subspaces S_l and S_{l+1} is small. I.e., for $x \in S_l$ and $y \in S_{l+1}$, $|\bar{G}(x) - \bar{G}(y)|$ is a polynomial of the binary string length n .
- **Wide-gap class.** In this case, there exists two subspaces S_k and S_{k+1} , the difference of the mean first hitting times between two subspaces S_k and S_{k+1} is large. I.e., for any $x \in S_k$ and $y \in S_{k+1}$, $|\bar{G}(x) - \bar{G}(y)|$ is an exponential of the binary string length n .

The role of population diversity can be reflected on the initial population. For wide-gap problems, without losing generality, we assume the wide gap is located between two subspaces S_k and S_{k+1} . Let's consider a special case of the wide-gap problems: for $l \leq k$ and $x \in S_l$, $\bar{G}(\xi_1 = x)$ is a polynomial of n and for $l > k$ and $x \in S_l$, $\bar{G}(\xi_1 = x)$ is an exponential of n .

Then there exists a superlinear scalable speedup when using the scalable PEAs. It can be drawn in the following. Assume the initial population is uniformly distributed on the population space $S(N)$. Let p be the probability for one individual in the initial population to satisfy $\bar{G}(x)$ being a polynomial of the n .

The mean first hitting time of PEA(1) satisfies:

$$\bar{G}_1 \geq (1-p)\text{expo}(n);$$

where $\text{expo}(n)$ is an exponential of n .

And the mean first hitting time of the PEA(N) satisfies:

$$\bar{G}_N \leq (1-p)^N \text{expo}(n),$$

where $\text{expo}(n)$ is an exponential of n .

Then we get a superlinear scalable speedup if p is greater than $1 - (1/N)^{1/N}$ for some N :

$$g_N \geq \frac{1}{(1-p)^N} > N.$$

However for some easy problem, e.g., the One-Max problem, such a uniform initialization (i.e., to increase the diversity of the initial population) does not lead to a superlinear speedup. This can be explained as follows: If for any two individuals x and y , the difference between their mean first hitting times $|\bar{G}(x) - \bar{G}(y)|$ is very small, then to increase the diversity in the initial population doesn't bring much more benefit. No superlinear g-speedup might happen.

As a consequence of the above analysis, the superlinear g-speedup can happen only if the first hitting times $\bar{G}(\xi_1 = x)$ have a large variance. This leads to a practical approach to check whether a superlinear speedup happens: to run PEA(1) N times, calculate the mean first hitting time $\bar{G} = \sum_x \bar{G}(x)/N$, and let K be the numbers of runs which the first hitting times less than $\bar{G}/N$ and calculate the ratio

$$\frac{1}{(1-K/N)^N} > N?$$

If the above inequality holds, then a superlinear scalable speedup might exist.

But if no diversity is in the initial population, the superlinear scalable speedup may don't happen even for the case of wide-gap problem.

Assume there is a wide gap between the subspaces S_k and S_{k+1} . We consider the PEA(1) first. Let q is the probability for PEA(1) jumping from the subspace S_{k+1} to the subspace S_k , then the mean first hitting time for PEA(1) to jump over the wide gap between S_k and S_{k+1} is $1/q = \text{expo}(n)$, where $\text{expo}(n)$ is an exponential of n .

Next we discuss the scalable PEA(N). Assume all individuals in the initial population take values from the subspace S_{k+1} , then for the scalable PEA(N), the probability for one individual jumping from the subspace S_{k+1} to the subspace S_k is no more than

$$1 - (1-q)^N < Nq,$$

then the first hitting times for the scalable PEA(N) to jump over the wide gap is not greater than $\text{expo}(n)/N$, which leads to sub-linear scalable speedup.

V. A CASE STUDY

At the moment, a theoretical analysis can be made only for those simple EAs and problems. For those classical optimization problems, we can observe the superlinear speedup from experiments. This section displays an experimental case study, which is used to demonstrate the relationship between the speedup and the population size.

The problem considered here is the subset sum problem, described as follows: given a set $W_n = \{w_1, \dots, w_n\}$ of n positives and a positive C , find a subset S of W such that the sum of the elements in S are closest to but not exceeding C . This problem is chosen here because of two reasons: first we can easily represent the solution by a binary string and define genetic operators; secondly, through setting different weights W_n , the instances of the problem can be tuned into different degrees of difficulty: from the extreme easy problem: the One-Max Problem to the intractable hard problem. A early analysis of EAs for solving the problem can be found in [8].

A solution S to the subset sum problem can be represented by a string $x = (s_1 \dots s_n)$ where $s_i \in \{0, 1\}$. The presence of w_i in S means that $s_i = 1$ while its absence is represented by $s_i = 0$. A feasible solution to the subset sum problem is a string $x = (s_1 \dots s_n)$, $s_i \in \{0, 1\}$, such that

$$\sum_{i=1}^n w_i s_i \leq C,$$

where $F(x) = \sum_{i=1}^n w_i s_i$ is called the objective function. The optimal solution is the string that maximizes the objective function (without exceeding C).

The fitness function can be defined as

$$f(x) = \begin{cases} F(x), & \text{if } x \text{ is a feasible solution;} \\ 0, & \text{else.} \end{cases}$$

In this section, we are interesting in three instances of subset sum problems, which will demonstrate different speedup. The first instance is the simplest, equivalent to the One-Max function,

$$w_i = 1, \quad i = 1, \dots, n, \\ C = \sum_{i=1}^n w_i.$$

It is obvious that $x = (1 \cdots 1)$ is the unique optimal solution and any subset of W_n is a feasible solution.

The second instance equals to the linear function

$$w_i = i, \quad i = 1, \dots, n, \\ C = \sum_{i=1}^n w_i.$$

The third one is

$$w_i = i, \quad i = 1, \dots, n, \\ C = \sum_{i=1}^n w_i / 2.$$

Now let's consider scalable PEA(N) using the following genetic operator.

- Mutation: Bitwise mutation with probability $1/n$.
- Selection: An elitism selection.

Two different selections are compared in the experiment:

- **S1**: an independent run of N copies of PEA(1), where in each PEA(1), it only accepts a better solution;
- **S2**: the winner-take-all selection.

The experiment setting is given as below: the population size is take from 1 to 50. The result is averaged over 200 independent runs. Since the maximum fitness values are known for these three instances, therefore the halting criterion is to let $f^* = f_{\max}$.

For the first instance, we already have made a theoretical analysis to it¹.

The following theorem gives the lower bound of the mean first hitting times.

Theorem 4: Let $\bar{G}_w$ be the worst-case mean first hitting time of PEA(N), using bitwise mutation and elitism selection, to find the optimal point. Then it is lower-bounded by

$$\bar{G}_w = \Omega\left(\max\left\{\frac{n \ln n}{N}, \frac{n \ln \ln N}{\ln N}, 1\right\}\right), \quad (22)$$

where N is the population size and n is the binary string length.

The theorem below presents that the PEA(N) using selection S2 can reach the above lower bound.

Theorem 5: Let $\bar{G}_w$ be the worst-case mean first hitting time of the PEA(N), using bitwise mutation and selection S2, to find the optimal point, then it is upper-bounded by

$$\bar{G}_w = O\left(\max\left\{\frac{n \ln n}{N}, \frac{n \ln \ln N}{\ln N}, 1\right\}\right), \quad (23)$$

where N is the population size and n is the binary string length.

¹Due to limitation of paper size, the proof of the theorem is not given here, but can be found in the paper "Analysis of First-hitting Times of Population-based EAs: (1) One-Max Problem", which has been submitted to IEEE Transactions on Evolutionary Computation

From the above two theorem, we see the speedup can be upper-bounded by

$$O\left(\min\left\{N, \frac{\ln n \ln N}{\ln \ln N}, n \ln n\right\}\right). \quad (24)$$

From the above theoretical speedup, we see that the scalable speedup is a function of N , no more than a linear speedup (when the binary string length n is fixed). Therefore, we cannot expect a superlinear speedup. The experimental result in Figure 1 and 2 demonstrate the scalable speedup with respect to the cases of the initial population satisfies the uniform distribution and the initial population satisfies the uniform distribution for the first instance.

Fig. 1. The experimental result of scalable speedup for the first instance, where the initial population satisfies the uniform distribution. S1 means the selection operator S1; S2 the selection operator S2. x the linear speedup.

Fig. 2. The experimental result of scalable speedup for the first instance, where the initial population starting from the worst point. S1 means the selection operator S1; S2 the selection operator S2. x the linear speedup.

We can see from these two figures,

- If the individual in the initial population satisfies the uniform distribution, the scalable speedup is no more than a linear speedup, where the linear speedup happens only when the population size is very small; this has confirmed the theoretical analysis.
- If the individual in the initial population takes the worst point, then the scalable speedup is less than a linear speedup even for a small population size.
- This confirms the diversity brings a faster speedup. The initial population in the first figure has a larger

population diversity than in the second figure, so the speedup is faster too.

The result in Figures 3 and 4 demonstrate the scalable speedup for the second instance with respect to the cases of the initial population satisfies the uniform distribution and the initial population satisfies the uniform distribution.

Fig. 3. The experimental result of scalable speedup for the second instance, where the initial population satisfies the uniform distribution. S1 means the selection operator S1; S2 the selection operator S2; x the linear speedup.

Fig. 4. The experimental result of scalable speedup for the second instance, where the initial population starting from the worst point. S1 means the selection operator S1; S2 the selection operator S2. x the linear speedup.

The following observations can be derived from these two figures:

- For both initial distributions, no superlinear speed have been found. Even no linear speedup can happens in this case.
- The speedup in the first figure is faster than that in the second one, because the initial population in the first figure has a larger population diversity than in the second one.

The result in Figures 5 and 6 demonstrate the scalable speedup for the third instance with respect to the cases of the initial population satisfies the uniform distribution and the initial population satisfies the uniform distribution.

Since this instance is much more difficult than the first two instances, so the superlinear scalable speedup appears.

- The result in Figure 5 demonstrate that a superlinear scalable speedup appears when the initial populations satisfies the uniform distribution.

Fig. 5. The experimental result of scalable speedup for the third instance, where the initial population satisfies the uniform distribution. S1 means the selection operator S1; S2 the selection operator S2. x the linear speedup.

Fig. 6. The experimental result of scalable speedup for the third instance, where the initial population starting from the worst point. S1 means the selection operator S1; S2 the selection operator S2. x the linear speedup.

- However, if the initial population loses the diversity, i.e., taking the worst point, then the result in Figure 6 shows that a scalable speedup never appear.

VI. CONCLUSION

Inherent parallelism is regarded as one of the most important advantages of evolutionary algorithms. This paper has made an initial study on the speedup of scalable PEAs and tried to explain how and why the superlinear speedup happens in scalable PEAs. We have shown the traditional definition of speedup is not suitable to describe the speedup in the scalable PEAs. Therefore a new definition of speedup, scalable speedup, is presented based on the first hitting time. The scalable speedup is tightly connected to the generation-based speedup (g-speedup). Such a g-speedup can be used to simplify the theoretical analysis of speedup without considering implementing details of PEAs.

In this paper we have found that the superlinear speedup may happen for some wide-gap problems, but it may not exist for some narrow-gap problems. The paper intent to show that population diversity is one of the crucial factors for the scalable PEAs to obtain a superlinear speedup when solving the wide-gap problem.

The paper also presents an experimental case study, which demonstrates the superlinear speedup can happen. For narrow-gap problems, since the variance of the first hitting

time among different subspaces is small, then the superlinear speedup cannot happen; for a wide-gap problem, the variance of the first hitting time is big, it is possible to generate a superlinear speedup. A straightforward conclusion is that scalable PEAs may be more efficient to solve hard problems, e.g., wide-gap problems, rather than easy problems, e.g., the One-Max Problem.

This is an initial application of the first hitting times on the speedup of scalable PEAs. In the future, we will analyze different instances of the subset sum problem and other classical combinatorial optimization problems. Like what we have done on the first instance, we need to obtain more rigorous bounds on the scalable speedup. We also are interested in looking for the theoretical conditions, which can be used to decide whether superlinear speedup can happen.

ACKNOWLEDGMENT

This work is supported by UK Engineering and Physical Research Council under Grant (EP/C520696/1) and National Natural Science Foundation of China under Grant (60443003).

REFERENCES

- [1] H. Mühlenbein, "Evolution in time and space-the parallel genetic algorithm," in *Proc. of the 1st Workshop on Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. Morgan Kaufmann, 1991, pp. 316–337.
- [2] E. Cantú-Paz, "Markov chain models of parallel genetic algorithms," *IEEE Trans. on Evolutionary Computation*, vol. 4, no. 3, pp. 216–226, 2000.
- [3] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.
- [4] E. Alba, "Parallel evolutionary algorithms can achieve super-linear performance," *Inf. Process. Lett.*, vol. 82, no. 1, pp. 7–13, 2002.
- [5] J. He and X. Yao, "From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 5, pp. 495–511, 2002.
- [6] R. Shonkwiler, "Parallel genetic algorithms," in *Proc. of the 5th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1993, pp. 199–205.
- [7] W. Rivera, "Scalable parallel genetic algorithms," *Artif. Intell. Rev.*, vol. 16, no. 2, pp. 153–168, 2001.
- [8] J. He and X. Yao, "Drift analysis and average time complexity of evolutionary algorithms," *Artificial Intelligence*, vol. 127, no. 1, pp. 57–85, 2001.
- [9] —, "Towards an analytic framework for analysing the computation time of evolutionary algorithms," *emphArtificial Intelligence*, vol. 145, no. 1-2, pp. 59–97, 2003.