ELSEVIER

Information Processing Letters 82 (2002) 7–13

Information Processing Letters

www.elsevier.com/locate/ipl

Parallel evolutionary algorithms can achieve super-linear performance

Enrique Alba

Dpto. de Lenguajes y Ciencias de la Computación, Univ. de Málaga, Campus de Teatinos (3.2.12), 29071 Málaga, Spain

Abstract

One of the main reasons for using parallel evolutionary algorithms (PEAs) is to obtain efficient algorithms with an execution time much lower than that of their sequential counterparts in order, e.g., to tackle more complex problems. This naturally leads to measuring the speedup of the PEA. PEAs have sometimes been reported to provide super-linear performances for different problems, parameterizations, and machines. Super-linear speedup means that using "m" processors leads to an algorithm that runs more than "m" times faster than the sequential version. However, reporting super-linear speedup is controversial, especially for the "traditional" research community, since some non-orthodox practices could be thought of being the cause for this result. Therefore, we begin by offering a taxonomy for speedup, in order to clarify what is being measured. Also, we analyze the sources for such a scenario in this paper. Finally, we study an assorted set of results. Our conclusion is that super-linear performance is possible for PEAs, theoretically and in practice, both in homogeneous and in heterogeneous parallel machines.

Keywords: Evolutionary algorithms; Parallel algorithms; Performance evaluation; Super-linear speedup; Serial fraction

1. Introduction

Evolutionary algorithms (EAs) are techniques for optimization and learning [6]. They are quite a large set of algorithmic families representing bio-inspired systems with a behavior drawn from Nature. In EAs, evolution is the basic force driving a population of tentative solutions towards problem regions where the optimal solutions are located. Since in this paper we will focus on parallel EAs (PEAs) in which many computers are used to speed up the search, let us begin with the description of a PEA. In Algorithm 1 we provide the pseudo-code.

```
node i: t := 0;
        initialize & evaluate [P(t)];
        while not stop_condition do
          P'(t) := variation [P(t)];
          evaluate [P'(t)];
          P''(t) := select [P'(t)];
          P'''(t) := communication[Δ_i, P''(t)]
          t := t + 1;
        end while
```

Algorithm 1. Parallel Evolutionary Algorithm.

A population $P(0)$ of data structures (usually strings or trees of symbols) is initially *generated* at random and evaluated to assess their individual quality as problem solutions (*fitness* values). A PEA repeats a loop in which the present population $P(t)$ undergoes a

E-mail address: eat@lcc.uma.es (E. Alba).

0020-0190/01/$ – see front matter
PII: S0020-0190(01)00281-2

variation phase by the application of some operators. Some popular operators are mate selection, crossing random slices between two individuals, and mutation of their contents (random changes). Then, a replacement step is performed in order to build up the new population $P(t+1)$ from the old one $P(t)$.

In a PEA, there are many nodes, each one performing this loop in parallel, with an additional phase of communication of the ith node with its neighboring set of nodes Δ_i, in which individuals or statistics are exchanged. Depending on the size of the population in each node, the number of nodes, and the frequency of the interactions, it is usual to distinguish between two types of PEAs: distributed (dEA) and cellular (cEA) PEA models [1,11]. A dEA has a small number of nodes, many tens of individuals in each node, and performs sporadic communications [20]. On the contrary, a cEA has a large number of nodes, usually a single individual in every node, and performs a tight and frequent communication with neighboring nodes [18]. Many hybrids also exist (see, e.g., [1]).

These two kinds of *structured* EAs have a different behavior than that of single-population (*panmictic*) ones, because they use decentralized reproduction. Both dEAs and cEAs directly suggest parallel execution in MIMD and SIMD computers, respectively. Usually, panmictic EAs are implemented as sequential algorithms, although they can also be executed in parallel, e.g., to perform evaluations in parallel [1].

Since measuring the performance of any PEA is important, some performance measures, such as *speedup*, have been borrowed from traditional algorithms. However, the definition of speedup has suffered some distortion when used in the EA field. Therefore, our first goal is to define speedup in the context of PEAs. It is also interesting to notice that many authors have reported super-linear speedups for different PEAs [2, 7,15]. Consequently, we would like to know whether super-linear speedup is a fact or only a consequence of a non-orthodox measure definition.

This paper is organized as follows. The next section revisits speedup and some related performance measures. In Section 3 we propose a new taxonomy of speedup measures. Section 4 studies whether super-linear speedup is possible in PEAs. Section 5 contains several brief PEA performance studies. Finally, some concluding remarks are outlined in Section 6.

2. Speedup and related performance measures

We denote by T_m the execution time for an algorithm using m processors. The PEA-adapted definition of speedup should then compute the ratio between the *mean* execution time on a uni-processor $E[T_1]$ and the *mean* execution time on m processors $E[T_m]$ because of its non-deterministic nature:

$$s_m = \frac{E[T_1]}{E[T_m]}. \tag{1}$$

With this definition we can distinguish amongst *sub-linear* speedup ($s_m < m$), *linear* speedup ($s_m = m$), and *super-linear* speedup ($s_m > m$). A related measure called *efficiency*, used to normalize the speedup value to a certain percentage (100% efficiency means linear speedup) is given by:

$$e_m = \frac{s_m}{m} \times 100\%. \tag{2}$$

Finally, Karp and Flatt [13] have devised an interesting metric for measuring the performance of any parallel algorithm that can help us to identify much more subtle effects than using speedup alone. They call it the *serial fraction* of the algorithm (f_m):

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m}. \tag{3}$$

As far as we know, this is the first attempt to use the serial fraction to analyze PEAs (see also [3]). Ideally, the serial fraction should stay constant for an algorithm. If a speedup value is small (e.g., efficiency around 87%), we can still say that the result is good if f_m remains constant for different values of m, since the loss of efficiency is due to the limited parallelism of the program. On the other side, smoothly increasing f_m is a warning that the granularity of the parallel tasks is too fine. A third scenario is possible in which a significant reduction in f_m occurs, indicating something akin to super-linear speedup. If super-linear speedup occurs, then f_m would take a negative value.

To justify why somebody should be interested in using the serial fraction measure to indicate super-linear values, first we must say that, since they are expressed by negative quantities of f_m, it is easy to notice for a reader. Second, using serial fraction allows us to link our results with other existing similar values for deterministic algorithms like the ones explained in [13,14] and in many interesting

references of the CALMA project documents [21]. Finally, as mentioned above, with f_m we can identify more subtle effects (trends) in the parallel program than when only using speedup.

3. A taxonomy of speedup measures for EC

A well-accepted way of measuring numerical performance is to check the number of evaluations of the objective function needed to locate the optimum. We call this the *numeric effort* of the algorithm. On the other hand, parallel performances can be measured in several ways, usually requiring computing the time needed to locate a solution. Since our focus is on speedup, we start by giving a taxonomy of speedup measures (see Table 1).

Type I (*strong* speedup) is provided to emphasize the fact that speedup must compare the PEA run time with the best-so-far sequential algorithm, whether of evolutionary nature or not. Because of the difficulty of finding the current most efficient algorithm for tackling the problem at hand, most designers of parallel algorithms (including PEAs) have not used strong speedup. Instead, they use a *weak* measure in which they compare the parallel algorithm with their own sequential version of their algorithm (type II in Table 1). Also, it could be possible to have a second (weak) comparison against the best known EA solution, but this would be as difficult to achieve in practice as comparing it against the best-so-far known algorithm. Here, we use the words "sequential" and "parallel" to stand for uni- and multi-processor executions, respectively.

We could run the algorithms with an a priori stopping criterion based, e.g., in a maximum number of evaluations, but this practice allows the researcher to get any desired and arbitrary speedup value by deciding upon the number of steps of the PEA. So we consider this measure (type II.B) undesirable for studying speedup. It might be useful for other purposes, e.g., to compare the final error of the best solution given by the sequential EA against the one given by the parallel algorithm. But, for measuring speedup, we think that the same target *solution quality* should be defined for both the sequential and the parallel EA, and only then we will be allowed to compare their run times (type II.A).

Table 1
Taxonomy of speedup measures

I. Strong speedup
II. Weak speedup
A. Speedup with solution-stop
1. Versus panmixia
2. Orthodox
B. Speedup with predefined effort

In fact, we could even decide to compare the PEA on m processors against a panmictic—single—node (type II.A.1). But if so, we should be comparing two clearly different algorithms. Consequently, it seems more appropriate to define a measure in which the same PEA is run on 1 and m processors (type II.A.2), and then to compute their average time ratio. Of course, it is assumed that the code executed in 1 and m processors is the same, and that no additional code is artificially or unnecessarily executed in sequential that could inflate its execution time.

In summary, speedup must be clarified in the PEA field, specially relating the base sequential algorithm for comparisons. First, we must use *average* and not absolute times, due to the non-deterministic PEA behavior. Second, the uni and multi-processor *implementations* should be *exactly the same*, thus avoiding the use of a panmictic EA in the uni-processor case versus a decentralized one in the multi-processor case (as pointed out in [16]). Third, for a meaningful speedup, we must run the PEA until a *solution* for the problem is found, as strongly claimed by some authors [3,8, 12]. With these requirements, speedup will provide a useful value indicating the advantage of adding more processors; otherwise, it could be misleading.

4. Is it really possible to have super-linear speedup in PEAs?

A number of authors have analyzed PEAs by considering different criteria, and many of them came out with a super-linear speedup result when using a parallel machine. See super-linear performances with parallel genetic algorithms (PGAs) in [2,3,7,15] and with parallel genetic programming (PGP) in [5].

Speedup has been studied in traditional parallel algorithms for many years, especially for homogeneous

workstation clusters and multi-computers. However, we also consider here an open research line consisting of studying the speedup for heterogeneous computers. In this last case, it seems more appropriate that performance figures use wall clock time rather than CPU time, since there is an obvious problem with defining a reference point against which the results should be compared [21]. The reference point should be the execution time on the fastest single processor [10]. Using this measure, we can at least assume that a speedup greater than unity implies that the parallel system is faster than the uni-processor. In [10], Donaldson et al. have shown that there is no theoretical upper limit for the speedup in heterogeneous systems, which can explain a super-linear value in this case.

On the other hand, based on our own experience with PGAs, we can expect to get super-linear speedups sometimes. But which are the sources behind the super-linear speedup in PEAs? We propose a classification into *implementation*, *numerical*, and *physical* sources:

#1. Implementation source. The algorithm being run on one processor is "inefficient" in some way. For example, if the sequential algorithm uses linear lists of data, the parallel one is faster because it manages shorter lists and merges the results. Super-linear speedup would not occur in this last case if the sequential algorithm were to use a binary data tree, for example, [21]. On the other hand, the complexity of the operators (selection, crossover) reduces dramatically when the population is split and the resulting chunks are dealt with in parallel. If the sequential algorithm is panmictic, then the parallel one most probably will exhibit a super-linear performance since its parallel decentralized execution has two additional advantages: sources #2 and #3 (discussed later). In the case of heterogeneous computers, using different compilers (e.g., different Java interpreters) can also be a source for super-linear speedup.

#2. Numerical source. Since the search space is usually very large, the sequential program may have to search a large portion before finding the required solution. On the other hand, the parallel version may find the solution more quickly due to the change in the order in which the space is searched. This holds for PEAs as well as for other algorithms such as Branch-and-Bound [14]. In fact, heuristics have a non-zero probability of finding a solution after T seconds, for any $T > 0$. By running an algorithm on several processors instead of just one we increase the probability of finding a solution. This has been proven by Shonkwiler [17], where we can even find a numerical model for characterizing the speedup value in PEAs: $s_m = m \cdot a^{m-1}$ (super-linear speedup when $a > 1$).

#3. Physical source. When moving from a sequential to a parallel machine, it is often the case that one gets more than an increase in CPU power. Other resources, such as memory, caché, etc. may also increase linearly with the number of processors. The parallel algorithm may be able to achieve super-linear speedup by taking advantage of these additional resources. In heterogeneous clusters the resources are different from each other, with unpredictable results.

We therefore conclude that super-linear speedup is possible in PEAs, as well as for other algorithms, theoretically and as a result in empirical tests, both for homogeneous [3,21] and heterogeneous [4,10] computing networks. For certain PGA applications it could be even common.

5. Examples of super-linear performance with parallel GAs

In order to offer a complete view of the super-linear speedup scenarios we now turn to give various examples of what is meant by super-linear speedup. The PEA we are using is a distributed GA in which every island performs a steady-state GA [19]. Each one of the 8 steady-state islands has $\mu = 64$ individuals and creates one individual $(\mu + 1)$-EA in every step. This is done by applying the selection (fitness proportional for homogeneous, and binary tournament for heterogeneous), two point crossover ($P_c = 1.0$ and $P_c = 0.85$ for homo/hetero cases), bit-flip mutation (always $P_m = 1/\text{string\_length}$), and replacement of the worst only if the new one is better than it [2]. The islands are located in a unidirectional ring, each sending one copy of a random string every $(32 \cdot \text{popsize})/8$ evaluations. The target island incorporates it into its population if it is better than its presently worst string.

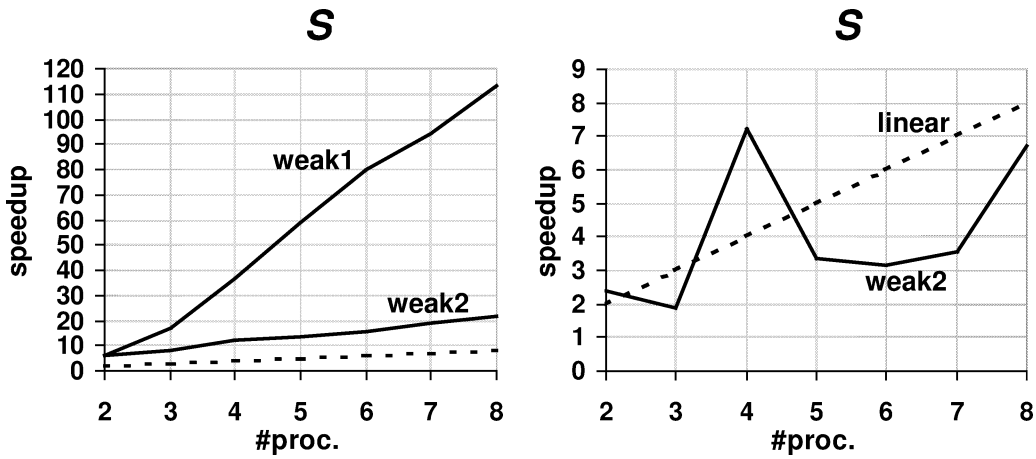

Fig. 1. Speedup for SPH16-32 (left) and SSS128 (right) in a homogeneous cluster.

The reception of incoming strings is asynchronous for efficiency [2,13].

We show an assorted (and brief) set of results for homogeneous as well as for heterogeneous machines to offer a non-biased and wide spectrum of possible scenarios. Let us begin with the homogeneous case. In the left graph of Fig. 1, we show the results on the problem of maximizing the square sum of 16 variables, each one encoded in 32 bits (SPH16-32). The dashed line represents linear speedup. We can see that the unfair comparison against a panmictic steady-state GA (*weak*1 ≡ type II.A.1) can yield huge values of super-linear speedup, since the distributed GA is parallel and needs a lower number of evaluations (not shown here due to space constraints). The line labeled *weak*2 (type II.A.2) proves that super-linear orthodox speedup occurs with moderate figures.

But, although *weak*2 seems appropriate, this does not always mean to be predictable. In the right graph of Fig. 1, we plot the result of measuring orthodox speedup (*weak*2) on a combinatorial problem known as the subset sum problem with 128 integers (SSS128) [2]. Although there are irregular dependencies (jumps) due to the influences of the mapping islands-to-processors, it is clear in these tests that super-linear performance can be achieved (e.g., with 4 processors).

As to the heterogeneous case, Table 2 shows the results obtained with the problem of maximizing the number of ones in a 512 bit string. We use 4 Pentiums III NT at 550 MHz, one IRIX R10000 at 250 MHz, one LINUX Pentium III at 550 MHz, and two Digital AlphaServer processors at 300 MHz (in the given order) to hold the ring of sub-algorithms. They all are linked by a Fast Ethernet LAN (100 Mbps).

Table 2
Speedup for ONEMAX in homogeneous and heterogeneous systems

Computers	*m* (#processors)	Time (s)	Speedup	Serial fraction
NT-cluster	1	309.9		
NT-cluster	8	38.5	8.051	−0.001
LINUX	1	116.2		
Heterogeneous	8	21.3	5.435	0.067

First, notice in Table 2 that comparing the homogeneous configuration of 1 versus 8 NT workstations yields a slightly super-linear speedup (8.051). On the other hand, since the LINUX machine was the fastest uni-processor we use it for the speedup in the heterogeneous case [10]. To our surprise, the heterogeneous configuration scored a smaller absolute average time (21.3 s) than the NT homogeneous case (38.5 s). While the heterogeneous speedup is small (5.435), it is a good value since the serial fraction is very small (0.067). We have just begun to test heterogeneous cases and have no super-linear results until now, although they are possible in theory [10].

Table 3
Speedup for P-PEAKS in a heterogeneous system

Computers	m (#processors)	Time (s)	Speedup	Serial fraction
NT-cluster	1	2425.4		
NT-cluster	8	300.0	8.083	−0.001
Heterogeneous	8	316.8	7.655	0.006

Table 3 shows results obtained with a multimodal problem generator [9] we call P-PEAKS from which we have constructed a highly epistatic instance (512 bits and 512 peaks). This time, the NT-cluster of Pentium III with NT scored the overall best results, both in uni and multi-processor homogeneous configuration, again with a slightly super-linear speedup (negative serial fraction). The heterogeneous configuration again is good with a near-linear speedup (7.655) and a very small serial fraction that ensures us to have made a good parallelization of the algorithm.

With respect to heterogeneity, the lesson is twofold. First, super-linear speedup is theoretically possible in a heterogeneous computing system, although it might be hard to get such a result in practice. Second, we can take advantage from merging existing machines with new ones instead of buying a whole set of new workstations (good news!).

6. Concluding remarks

In this work we have discussed the speedup of parallel evolutionary algorithms. We have proposed a taxonomy of speedup measures and have identified the sources for a super-linear performance scenario. In addition, we have shown that super-linear speedup is possible, both in theory and practice, for homogeneous as well as heterogeneous computational resources. Finally, we have provided concrete results in order to show different aspects of the speedup of PEAs.

We have been experiencing super-linear speedups for many configurations and problems. Although super-linear speedup is always controversial, it is a fact. Our future work will address other parallel machines, algorithms, and problems to better characterize speedup in PEAs.

References

[1] E. Alba, J.M. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (4) (1999) 31–52.
[2] E. Alba, J.M. Troya, Analyzing synchronous and asynchronous parallel distributed genetic algorithms, Future Generation Comput. Systems 17 (2001) 451–465.
[3] E. Alba, J.M. Troya, Measuring the speedup of parallel genetic algorithms, J. Parallel Distributed Comput., 1999, submitted.
[4] E. Alba, A.J. Nebro, J.M. Troya, Heterogeneous computing and parallel genetic algorithms, J. Parallel Distributed Comput., to appear.
[5] D. Andre, J.R. Koza, A parallel implementation of genetic programming that achieves super-linear performance, J. Inform. Sci. 106 (3–4) (1998) 201–218.
[6] T. Bäck, D. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford University Press, Oxford, 1997.
[7] T.C. Belding, The distributed genetic algorithm revisited, in: L.J. Eshelman (Ed.), Proc. 6th International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1995, pp. 114–121.
[8] E. Cantú-Paz, D.E. Goldberg, Predicting speedups of idealized bounding cases of parallel genetic algorithms, in: T. Bäck (Ed.), Proc. 7th International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1997, pp. 113–120.
[9] K.A. De Jong, M.A. Potter, W.M. Spears, Using problem generators to explore the effects of epistasis, in: T. Bäck (Ed.), Proc. 7th International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1997, pp. 338–345.
[10] V. Donaldson, F. Berman, R. Paturi, Program speedup in a heterogeneous computing network, J. Parallel Distributed Comput. 21 (1994) 316–322.
[11] V.S. Gordon, D. Whitley, Serial and parallel GAs as function optimizers, in: S. Forrest (Ed.), Proc. 5th International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1993, pp. 177–183.
[12] W.E. Hart, S. Baden, R.K. Belew, S. Kohn, Analysis of the numerical effects of parallelism on a parallel genetic algorithm, in: Proc. of IEEE Workshop on Solving Comb. Opt. Problems in Parallel, CD-ROM IPPS97, 1997.
[13] A.H. Karp, H.P. Flatt, Measuring parallel processor performance, Comm. ACM 33 (5) (1990) 539–543.
[14] T. Lai, S. Sahni, Anomalies in parallel brach-and-bound algorithms, Comm. ACM 27 (6) (1984) 594–602.
[15] S.C. Lin, W.F. Punch III, E.D. Goodman, Coarse-grain parallel genetic algorithms: Categorization and new approach, in: Proc. 6th IEEE Parallel Distributed Processing, 1994, pp. 28–37.
[16] W.F. Punch, How effective are multiple populations in genetic programming, in: Proceedings of the Genetic Programming Conference, Morgan Kaufmann, Los Altos, CA, 1998, pp. 308–313.
[17] R. Shonkwiler, Parallel genetic algorithms, in: S. Forrest (Ed.), Proc. 5th International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1993, pp. 199–205.
[18] P. Spiessens, B. Manderick, A massively parallel genetic algorithm, in: R.K. Belew, L.B. Booker (Eds.), Proc. 4th

International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991, pp. 279–286.
[19] G. Syswerda, A study of reproduction in generational and steady-state genetic algorithms, in: G. Rawlins (Ed.), Foundations of Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991, pp. 94–101.
[20] R. Tanese, Distributed genetic algorithms, in: J.D. Schaffer (Ed.), Proc. 3rd International Conference on Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1989, pp. 434–439.
[21] UEA CALMA Group, CALMA Project Report 2.4: Parallelism in combinatorial optimisation, Technical Report, SCI-University of East Anglia, UK, September 18, 1995.