Research paper

Struct Multidisc Optim 28, 340–348 (2004)
DOI 10.1007/s00158-004-0412-1

Design optimization of discrete structural systems using MPI-enabled genetic algorithm

S.D. Rajan and D.T. Nguyen

Abstract The focus of this paper is on the development and implementation of a genetic algorithm (GA)-based software system using message passing interface (MPI) protocol and library. A customized and improved form of simple GA used in previous research (Chen *et al.* 1997; Chen and Rajan 1998, 2000; Rajan *et al.* 1999) is parallelized. This MPI-enabled version is used to find the solution to finite element-based design optimization problems in a network of workstations. Results show that an almost linear speedup is obtained on homogenous hardware cluster and, with a proper load-balancing strategy, on heterogeneous hardware cluster.

Key words genetic algorithm, load balancing, MPI, parallel processing, structural optimization

1 Introduction

Genetic algorithms (GA) have evolved over the last three decades to be recognized as a very powerful tool in obtaining solutions to nonengineering and engineering design optimization problems. The simple GA, while powerful, is perhaps too general to be efficient and robust for structural design problems. First, function (or fitness) evaluations are computationally expensive because they typically involve finite element analysis. Second, the (feasible) design space is at times disjointed with multiple local minima. Third, the design space can be a function of boolean, discrete and continuous design variables. The use of GA to find the optimal solution(s) of engineering design problems is still an open research area. Experience with GA has indicated that more often than not, tuning the GA strategy and parameters can lead to a more efficient solution process for a class of problems.

One of the most interesting aspects of GA is the explosive growth in the number of strategies explored by researchers in a multitude of disciplines. There are two popular approaches to configuring the system when parallel processing is combined with GA.

(1) Master–slave model: This is the most popular approach. The master process controls the program flow by assigning tasks to the slave processes. In the context of GA, typically the slave processes compute the fitness values, while the master process uses this information to create the next generation.
(2) Island-migration model: In this model, the entire population is divided into subpopulations (or islands) that are associated with different processors. As the disparate populations evolve, periodically, the best individuals are exchanged between the subpopulations (migration).

Both these approaches have been used successfully in solving a variety of problems. In a recent report developed by a NASA–Langley committee chaired by Biedron *et al.* (1999), coarse-grained parallelism is touted as a cost-effective and efficient method for search techniques such as GA, response surfaces and neural nets. Next, we present a review of a fraction of the number of GA variations used in advancing the applicability of GAs, especially when parallel computing is available.

Eby *et al.* (1997) present an approach to optimal design of elastic flywheels using an injection island genetic algorithm (iiGA) that is a variation of the island-migration model. An iiGA in combination with a finite-element code is used to search for shape variations to optimize the specific energy density (SED) of elastic flywheels that is defined as the amount of rotational energy stored per unit mass. iiGAs seek solutions simultaneously at different levels of refinement of the problem representation (and correspondingly different definitions of the fitness function) in separate subpopulations (islands). Solutions are sought first at low levels of refinement with an ax-

Received: 29 May 2003
Revised manuscript received: 10 February 2004
Published online: 8 July 2004

S.D. Rajan[1, ✉] and D.T. Nguyen[2]

[1] Department of Civil Engineering, Arizona State University, Tempe, AZ 85287, USA
e-mail: s.rajan@asu.edu
[2] Department of Civil Engineering, Old Dominion University, Norfolk, VA 23539, USA

isymmetric plane stress finite element code for high-speed exploration of the coarse design space. Next, individuals are injected into populations with a higher level of resolution that uses an axisymmetric three-dimensional finite element model to fine-tune the flywheel designs. In true multiobjective optimization, various subfitness functions can be defined that represent good aspects of the overall fitness function. Allowing subpopulations to explore different regions of the fitness space simultaneously allows relatively robust and efficient exploration in problems for which fitness evaluations are costly.

Miki *et al.* (1999) present another variation of the island-migration model. A migration scheme that moves some individuals in one island to another island is adopted to create new population mixes. In the numerical examples, different values of the mutation rate and the crossover rate are assigned to different islands, thereby creating different GA environments in each of these islands. The optimization problem that is solved is the minimization of the volume of truss structures under tensile, buckling and displacement constraints. Other similar approaches can be found in Chipperfield and Fleming (1996).

Sarma and Adeli (2001) use a variation of the master–slave model. They employ a bilevel strategy in finding the solution to the design problem. First, parallel fuzzy GAs are used to obtain a continuous-variable minimum weight design. In this stage, the objective function and the constraints are considered to be fuzzy and a genetic search is performed with a preemptive constraint-violation strategy. Small constraint violations are allowed. This solution is then used as a preliminary startup design for the subsequent fuzzy discrete multicriteria cost optimization. Both OpenMP directives and message passing interface (MPI) calls are used in a shared memory data parallel computing and message passing distributed computing to take advantage of the best of both approaches.

Finally, an interesting approach is presented by Scott *et al.* (1995), where a hardware-based GA solution methodology is used. Speedups of 1–3 orders of magnitude are observed when frequently used software routines were implemented in hardware by way of reprogrammable field-programmable gate arrays (FPGAs). The prototype system uses VHDL to allow for easy scalability. It is designed to act as a coprocessor with the CPU of a PC. The user programs the FPGAs, which implement the function to be optimized. In simple tests, the prototype took about 6% as many clock cycles to run as the software-based GA. The authors suggest improvements that could realistically make the hardware system 2–3 orders of magnitude faster than the software-based GA.

We have two major foci or objectives in this paper. First, very briefly, the proposed improvements to the simple GA are discussed. These improvements are aimed at improving the reliability and efficiency of the overall process. Second, the major focus is on the development and implementation of a load-balancing methodology. The load-balancing issue is tackled so that the overall methodology works efficiently on both homogenous and heterogeneous computer clusters. A homogenous cluster is defined as one having identical computers connected by a switch, whereas a heterogeneous cluster is one where the computer hardware is not identical.

2 Genetic algorithm

The design problem can be stated as follows:

$$\text{Find } \mathbf{x} = \left\lfloor {}^{b}x_1, \ldots, {}^{b}x_{n_b}; {}^{i}x_1, \ldots, {}^{i}x_{n_d}; {}^{s}x_1, \ldots, {}^{s}x_{n_s} \right\rfloor$$

$$\text{to minimize } f(\mathbf{x})$$

$$\text{subject to } g_i(\mathbf{x}) \le 0 \quad i = 1, \ldots, n_i$$

$$h_j(\mathbf{x}) = 0 \quad j = 1, \ldots, n_e$$

$${}^{b}x_p \in \{0, 1\} \quad p = 1, \ldots, n_b$$

$${}^{i}x_q \in \left\{x_q^1, x_q^2, \ldots, x_q^{nq}\right\} \quad q = 1, \ldots, n_d$$

$${}^{s}x_r^L \le {}^{s}x_r \le {}^{s}x_r^U \quad r = 1, \ldots, n_s\,, \tag{1}$$

where $\mathbf{x}$ is the design variable vector, $f(\mathbf{x})$ is the objective function, n_i is the number of inequality constraints, n_e is the number of equality constraints, n_b is the number of boolean design variables, n_d is the number of discrete design variables selected from a list of nq values, and n_s is the number of continuous design variables. The genetic algorithm used in this research has evolved and been refined over time. In this section, we discuss some of the improvements that have been made to a simple GA in order to improve its overall performance. Further details can be found in prior publications (Chen *et al.* 1997; Chen and Rajan 1998, 2000; Rajan *et al.* 1999).

(1) *Adaptive penalty function:* GAs were developed to solve unconstrained optimization problems. However, engineering design problems are usually constrained. They are solved by transforming the problem to an unconstrained problem. The transformation is not unique and one possibility is to use the following strategy:

$$\text{minimize: } f(\mathbf{x}) + \sum_i c_i \times \max(0, g_i) + \sum_j c_j \times |h_j|\,, \tag{2}$$

where c_i and c_j are penalty parameters used with inequality and equality constraints. Determining the appropriate penalty weights c_i and c_j is always problematic. We use an algorithm here, where the penalty weight is computed automatically based on the traits of the current population and adjusted in an adaptive manner.

(2) *Improving crossover operators using the association string:* As discussed by some researchers, the one-point crossover is preferred for continuous domains

and the uniform crossover for discrete domains. However, schema representation still plays a pivotal role in the efficiency of the GA. If one uses a one-point crossover, then it is obvious that the ordering of the design variables is an important issue. Because the characteristic of one-point crossover is that the shorter schema has a better chance of surviving, if two variables that have less of an interdependency are placed adjacent to each other or two variables with a strong relationship are placed far away from each other, the crossover operation will make it more difficult for the GA to search the design space efficiently. To implement this strategy, we introduce an additional string called the *association string*. Results show that the association string improves the robustness of the solution process (Chen and Rajan 1998). To summarize, design variable ordering is important if the crossover point is static. If one-point crossover is used and the crossing point is randomly chosen along the string, then, theoretically, all design variables have equal probability of including the crossover point.

(3) *Mating pool selection:* The selection scheme (for generating the mating pool) together with the penalty function dictate the probability of survival of each string. While it is very important to preserve the diversity in each generation, researchers have also found that sometimes it may be profitable to bias certain schema. However, results from most of the selection rules, like roulette wheel, depend heavily on the mapping of the fitness function. In this paper, the tournament selection is used. There are at least two reasons for this choice. First, tournament selection increases the probability of survival of better strings. Second, only the relative fitness values are relevant when comparing two strings. In other words, the selection depends on individual fitness rather than the ratio of fitness values. This is attractive because, in this research, the fitness value contains the penalty term and does not represent the true objective function.

(4) *Elitist approach:* Research has shown that the GA with the incorporation of the elitist approach can be more reliable and efficient than the ones without. This approach is used in the current research.

(5) *Population size and stopping criteria:* Generally speaking, the initial population should contain uniformly distributed alleles. By this it is meant that, if possible, no chromosome pattern should be missed. Each chromosome is represented by n bits, with each bit being either 1 or 0. If the distribution of 1's (or 0's) in each bit location is to be uniform, the initial population size should be at least n. During the evolution, it is expected that the chromosome converges to some special pattern with the 0–1 choice decided for n locations. Assume that the choice of each bit is independent of all the other bits. Because the population size is n in each generation, after every generation, from a statistical viewpoint, we can expect to learn about at least one bit. Ideally, then, after n generations, one can expect to learn about all the n bits forming the chromosome. However, because each bit is not independent of the others, more than n generations are perhaps necessary to obtain a good solution. This suggests that the population size and the number of generations should be *at least* n. Numerical experience in our previous work suggests that using population and generation size of $2n$ leads to acceptable results efficiently.

3 Parallel GA

The overall algorithm used in a GA-based design optimization problem is quite simple. The overall flow is shown in Fig. 1. For engineering design problems, from a computational viewpoint, the fitness evaluation is the most expensive step. Hence, it would be prudent to parallelize the fitness evaluation step.

There are different ways one could devise the algorithm for evaluating the fitness function in parallel. We will show an evolutionary process in the development of the algorithm.

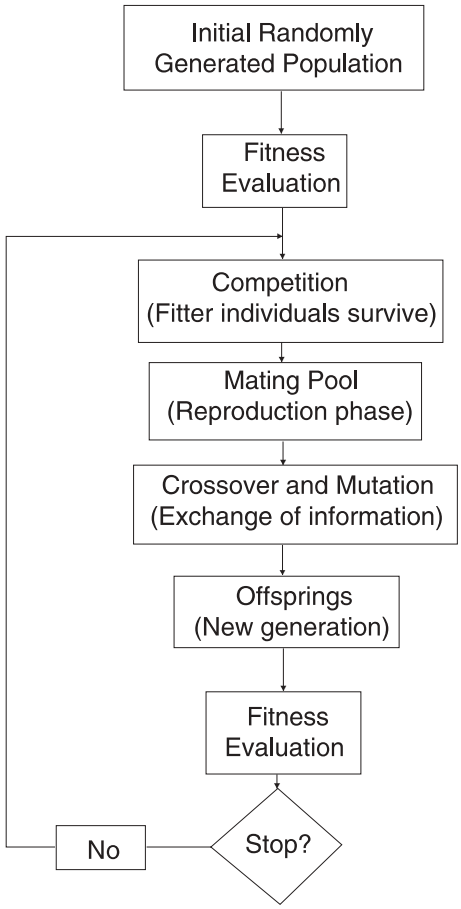

Fig. 1 Flow in a simple genetic algorithm (SGA)

Send all–then receive (SATR) Approach: We will assume that the number of available processes, n_p, is less than the population size, n_{pop}. In this approach, the master process divides the number of fitness evaluations equally amongst all the slave processes. It first sends the values of all the design variables associated with each individual (of the population) to the slave processes. After the information associated with the entire population is sent to the slave processes, the master process then waits to receive the values of the objective function and the maximum violation from all the slave processes. The detailed algorithm is presented below.

Master process

1. Set next available process as process $j = 1$.
2. Loop through all members of the population, $i = 1, 2, \ldots, n_{pop}$.
3. Generate the vector of design variables, x.
4. Pass this vector to process j.
5. Increment j. If $j = n_p$, set $j = 1$.
6. End loop.
7. Set next process as process $j = 1$.
8. Loop through all members of the population, i.
9. Receive the objective function value and the maximum constraint violation from process j.
10. Increment j. If $j = n_p$, set $j = 1$.
11. End loop.

Slave process

1. Set next available process as processor $j = 1$.
2. Loop through all members of the population, $i = 1, 2, \ldots, n_{pop}$.
3. If j is equal to the slave process number, receive the vector of design variables, compute and send the objective function value and the maximum constraint violation to the master process.
4. Increment j. If $j = n_p$, set $j = 1$.
5. End loop.

While this approach is straightforward, the problem with the above algorithm is that (a) for the master process, receives do not start until all the sends are completed, (b) for the slave process, sends do not start until all the receives are completed, and (c) load imbalance will take place in a heterogeneous computing environment. Under the SATR scenario, buffer overflows are likely to occur, and therefore, it is not used in the present study.

Load-balanced (LB) approach: Next, we present a modified approach where send and receive take place one after the other literally on demand. Initially, the master process instructs each slave process to evaluate the fitness associated with one member of the population by sending the values of the design variables for that member. Then it waits to receive the values of the objective function and the maximum violation from any of the slave processes (until the fitness evaluations of all the members of the population are completed). If there are more evaluations to be done, it passes the values of the design variables to that process. If all the evaluations are completed, it sends a message to that process that no more fitness evaluations are necessary.

Master process

1. Set number of evaluations completed, $n_{eval} = 0$.
2. Loop through $j = 1, \ldots, \min(n_{pop}, n_p)$.
3. Generate the vector of design variables, x. Pass this vector to process j. Increment n_{eval}.
4. End loop.
5. Loop through all members of the population, $i = 1, 2, \ldots, n_{pop}$.
6. Receive the objective function value and the maximum constraint violation from process j.
7. If $n_{eval} < n_{pop}$, generate the vector of design variables, x for member i. Pass this vector to process j. Increment n_{eval}. Else send no-more-evaluation message to process j.
8. End loop.

Slave process (valid only for process $j < n_{pop}$)

1. Loop until no-more-evaluation message is received.
2. Receive the vector of design variables, compute and send the objective function value and the maximum constraint violation to the master process.
3. End loop.

It should be noted that, in both the abovementioned approaches, only the master process executes the GA. Assuming that one integer word is 4 bytes, one double precision word is 8 bytes and the objective function and maximum constraint violation are designated as double precision, we can compute the total number of send and receive bytes as follows for every generation:

$$n_{send} = n_{receive} = 4n_{pop}\left(n_b + n_d + 2n_s\right) + 8\left(2n_{pop}\right). \quad (3)$$

Do-all load-balanced (DLB) approach: In the previous approaches, the GA is implemented and executed in the master process with only the fitness evaluations taking place in the slave processes. This approach requires that the values of the design variables be sent from the master process to the slaves. It should be noted that, as a fraction of the total program time, the time taken to execute the GA steps is a very tiny fraction. In this DLB approach, all the processes execute exactly the same program statements except for the part where the entire population is evaluated. As a byproduct, program maintenance is much easier because there is a single block of the program where process-related logic needs to be used.

Master process

1. Set number of evaluations completed, $n_{eval} = 0$.
2. Loop through $j = 1, \ldots, \min(n_{pop}, n_p)$.
3. Increment n_{eval}. Ask process j to evaluate member n_{eval} of the population.

4. End loop.
5. Loop through all members of the population, $i = 1, 2, \ldots, n_{pop}$.
6. Receive the objective function value and the maximum constraint violation from process j.
7. If $n_{eval} < n_{pop}$, increment n_{eval}, ask process j to evaluate member n_{eval} of the population. Else send no-more-evaluation message to process j.
8. End loop.
9. Broadcast the objective function and maximum constraint violation values for all the members of the population.

Slave process (valid only for process $j < n_{pop}$)

1. Loop until no-more-evaluation message is received.
2. Receive index of the member of the population whose objective function and constraints must be evaluated. Compute and send the objective function value and the maximum constraint violation to the master process.
3. End loop.
4. Receive the objective function and maximum constraint violation values for all the members of the population.

With this approach, the number of point-to-point send and receive bytes per generation is as follows:

$$n_{send} = n_{receive} = 20 n_{pop}\,, \tag{4a}$$

and the number of broadcast bytes is

$$n_{broadcast} = 16 n_{pop}\,. \tag{4b}$$

With this approach, the communication traffic is independent of the number and type of design variables.

4 Numerical examples

The focus of the current research is to develop and test a parallel, MPI-enabled GA for engineering problems. Hence, only a sizing optimal design problem is solved using an academic problem that has the desired characteristics (number of degrees of freedom and design variables) to test the parallel implementation. The results and conclusions, as we will discuss later, can be easily extended to other types of structural design problems, including solutions of more practical problems.

The sizing optimization problem is as follows:

$$\text{Find } \mathbf{x}_{k\times 1} \tag{5}$$

$$\text{to } \min f(\mathbf{x}) = \sum_{i=1}^{n} A_i L_i \rho_i \tag{6}$$

$$\text{subject to } g_i \equiv \sigma_i \le \sigma_a \quad i = 1, 2, \ldots, n \tag{7}$$

$$x_j^L \le x_j \le x_j^U \quad j = 1, 2, \ldots, k\,, \tag{8}$$

where $\mathbf{x}_{k\times 1}$ represents the cross-sectional areas of the truss members (design variables), $f(\mathbf{x})$ represents the mass of the truss (objective function) and g_i the stress constraints. To evaluate the fitness evaluation, one must compute the objective function and all the constraints. Without resorting to any approximation technique, a full finite-element analysis is required to evaluate the fitness as shown in (1).

Hardware: The numerical examples were generated on two different clusters. The first is labeled as high-cost cluster and is made up of more expensive computers and a high-performance switch. The second is labeled as a low-cost cluster and common off-the-shelf (COTS) computers and switch are used.

High-cost homogenous cluster information:
(a) Number of machines in the cluster = 7, (b) typical machine: Intel P4 1.7 GHz Dual Xeon, 512 MB RDRAM, Ultra 7200 rpm IDE Drive, Intel PRO/1000 T NIC, (c) Windows 2000 (SP 2), MPI-Softtech 1.6.3 (MPI Software Technology 2002), Cisco Catalyst 3550-12T switch.

Low-cost heterogeneous cluster information:
(a) Number of machines in the cluster = 2; (b) computer 1: Intel Dual P3-866 MHz, 768 MB RDRAM, Ultra 7200 rpm IDE Drive, 3COM 3C920 NIC; computer 2: AMD 1.2 GHz Athlon, 512 MB SDRAM, Ultra 7200 rpm IDE Drive, 3COM 3C920 NIC; (c) Windows 2000 (SP 2), MPI-Softtech 1.6.3, Linksys BEFSR41 10/100 Router.

A note about the terminology used in the numerical results. MPI allows multiple processes to be launched on a single computer system (or node). In the computer runs discussed below, an additional process is launched on the first machine and acts as the master process. When the number of processes is greater than 7, both the processors (CPU) on every computer were used in the high-cost cluster. In this case, we have distributed as well as shared memory scenarios. Similarly, when the number of processes is less than or equal to 7, only one processor of the dual-processor machine was used in the high-cost cluster. When the numerical examples were executed, both these clusters were converted to stand-alone clusters. By disconnecting the switch from the outside network, network traffic was restricted to just the machines connected to the switches. In addition, services not required for the MPI runs were stopped.

Cluster performance analysis: To better understand the performance issues with the two clusters, a special program was developed and executed on the two clusters. A vector containing floating point values was used to send messages of lengths varying between 4000 bytes (1000 words) and 4 000 000 bytes (1 000 000 words). These messages were

(a) broadcast from the master process to all the slave processes using MPI_BCAST, and
(b) sent from the master process to the slave processes one at a time using MPI_SEND and received using MPI_RECV.

The transmission rates from the two schemes are identified in Tables 1 and 2 as broadcast rates (BR) and point-to-point communication rates (PPCR), respectively.

High-cost homogenous cluster performance: The results are shown in Table 1.

Table 1 High-cost cluster performance

Number of processes (machines)	Broadcast rates (MB/s)	Point-to-point communication rates (MB/s)
2 (2)	16.8 to 49.2	17.4 to 49.2
3 (3)	9.8 to 24.6	8.0 to 23.0
4 (4)	2.6 to 24.5	2.5 to 15.7
5 (5)	0.95 to 17.6	1.4 to 11.9
6 (6)	0.84 to 17.3	1.1 to 9.8
7 (7)	0.66 to 15.3	0.56 to 9.0
10 (5)	0.4 to 15.9	0.41 to 9.6
14 (7)	0.012 to 14.2	0.012 to 3.8

Low-cost cluster performance: The results are shown in Table 2.

Table 2 Low-cost cluster performance

Number of processes (machines)	Broadcast rates (MB/s)	Point-to-point communication rates (MB/s)
2 (2)	1.44 to 9.5	1.53 to 8.9

The results not only show a wide variation within a column but also show a wide variation within a cell in the table. The latter variation is due to varying message lengths. Typically, one would expect the communication rate to increase with increasing message length and then start to decrease when the message length becomes very large. This monotonic behavior is rarely achieved in practice. However, the following conclusions can be drawn from the results:

(a) The communication rate is a function of the message length. While there is an optimal message length for a specific cluster, it can rarely be used in practice. One must, however, attempt to reduce not only the message length but also the number of times messages are sent and received to increase the efficiency of a parallel algorithm.
(b) Both BR and PPCR become less efficient with increasing numbers of processes and machines. It should be noted that, when more than one process is launched on a machine, SMP clusters can provide high performance if the shared memory bus of their nodes provide high performance. However, the motherboard on the High-Cost Cluster is not designed for high performance and does become a bottleneck.
(c) While BR and PPCR show similar performances with a small number of processes, BR is more efficient when the number of processes begins to increase. However, one must recognize that, due to (a), BR may not be very efficient unless optimal message lengths are used. Once again, due to the size of the cluster, the high-cost cluster is affected but not the low-cost cluster.

Test problems: The structural system that is designed is shown in Fig. 2. The planar truss is described in terms of two parameters – the number of bays and the number of storeys. The truss members are grouped into three groups per storey – horizontal members, vertical members and diagonal members. Hence, the number of design variables (cross-sectional areas) is equal to three times the number of storeys. The optimization problem is to find the optimal values of the cross-sectional areas of the members so as to minimize the mass of the truss subject to an axial stress constraint. The specific values used in the following examples are as follows:

$\rho = 0.00881448\,\text{lbm/in}^3$ (mass density)

$\sigma_a = 10\,000\,\text{psi}$ (allowable axial stress value)

$x_j^L = 0.1\,\text{in}^2$ and $x_j^U = 20\,\text{in}^2$ and precision is taken as $0.1\,\text{in}^2$

bay width $= 240\,\text{in}$

storey height $= 120\,\text{in}$

applied load, $P = 10\,000\,\text{lb}$

Two test problems are solved and are identified as TRUSS1 and TRUSS2. TRUSS1 is a structurally larger problem, meaning that one complete finite element analysis takes more time compared with the other model. How-

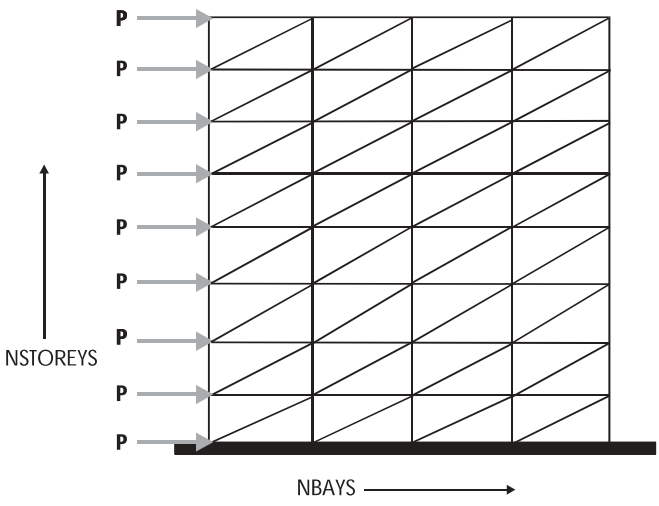

Fig. 2 Layout of the planar truss

ever, the number of design variables is smaller. TRUSS2, on the other hand, is different – a smaller structural model but with a much larger number of design variables. The problem details and results are presented next. All timing information is in terms of wall clock (or elapsed) time obtained using the MPI_WTIME function and represents the average value of two runs. It should be noted that wall clock time may vary as much as 10–20% between different runs of the same program and problem. A fitness-improvement tolerance could have been used as the convergence criterion. However, both these problems were executed for a fixed number of generations because the primary intent was to compare the different parallel GA implementations.

For a homogenous system, we compute the speedup as follows:

$$\text{speedup obtained for } n \text{ processes} = \frac{\text{Time for one process}}{\text{Time for } n \text{ processes}} \tag{9}$$

For a heterogeneous system, we compute the speedup as follows. Let the relative speed of the n_p processes with respect to the slowest process be denoted as $s_1, s_2, \ldots, 1, \ldots, s_{n_p}$ (with 1 corresponding to the slowest process), the time taken for each single process run be denoted as $t_1, t_2, \ldots, t_i, \ldots, t_{n_p}$ and t be the total time taken when n processes are used. Then

$$s_j = \frac{\max(t_1, t_2, \ldots, t_{n_p})}{t_j}, \tag{10}$$

$$\text{speedup obtained for } n \text{ processes} = \frac{\dfrac{1}{\sum_{j=1}^{n_p} s_j}\left(\sum_{i=1}^{n_p} s_i t_i\right)}{t}. \tag{11}$$

Test problem 1 (ID: TRUSS1): The problem-specific data are as follows:

Number of bays	300
Number of storeys	10
Nodes	3311
Elements	9010
Number of design variables	30
Chromosome Length	240
# of generations	100
# of function evals/generation	480

The initial population is randomly generated and the GA is terminated after 100 generations. The problem is solved in both the low-cost and high-cost clusters. The results are shown in Table 3(a) and 3(b). The initial objective function is 115 545 lbm and the final value after 100 generations is 23 475 lbm.

Remark 1. The timing values for the LB approach show that the program execution scales almost linearly in the high-cost (homogenous) cluster. There is a modest drop in efficiency as the number of processes is increased. Using (3), 245 760 bytes are sent and received every generation. With the DLB approach, the speedup once again is almost linear. Using (4a) and (4b), 9600 point-to-point bytes and 7680 broadcast bytes are sent and received every generation. However, there is no perceptible difference in efficiency between the two approaches. As discussed in the cluster performance analysis, this lack of difference is probably due to the fact that communication time is not significant for this problem.

Table 3 Results of TRUSS1 problem

(a) High-cost cluster

Number of processes (machines)	LB version Time (s)	LB version Speedup	DLB version Time (s)	DLB version Speedup
1 (1)	2178	1.0	2178	1.0
3 (2)	1092	1.99	1100	1.98
4 (3)	734	2.97	740	2.94
5 (4)	553	3.94	552	3.94
6 (5)	448	4.86	443	4.92
7 (6)	382	5.70	375	5.81

(b) Low-cost cluster

Number of processes (machines)	LB version Time (s) C1/C2[1]	LB version Speedup	DLB version Time (s) C1/C2	DLB version Speedup
1 (1)	5080/3306	1.0	5080/3306	1.0
3 (2)	2042	1.96	1997	2.0

[1] C1/C2: computer 1 and computer 2

The low-cost heterogeneous cluster shows an almost linear speedup, indicating that the approach of distributing the fitness evaluations works as intended – the number of fitness evaluations per generation computed by C1 and C2, on an average, is 190 and 290, respectively (a ratio of 1.53, which is the same as the ratio 5080/3306). The DLB approach is more efficient, showing an ideal speedup because the communication traffic is much less compared with the LB approach.

The design history is shown in Fig. 3.

Test problem 2 (ID: TRUSS2): The problem-specific data are as follows:

Number of bays	30
Number of storeys	50
Nodes	1581
Elements	4550
Number of design variables	150
Chromosome Length	1200
# of generations	50
# of function evals/generation	2400

The initial population is randomly generated and the GA is terminated after 50 generations. The results are

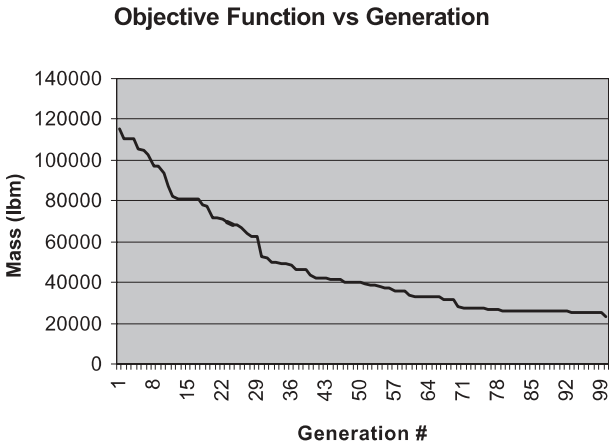

Fig. 3 Design history showing objective function versus generation (TRUSS1)

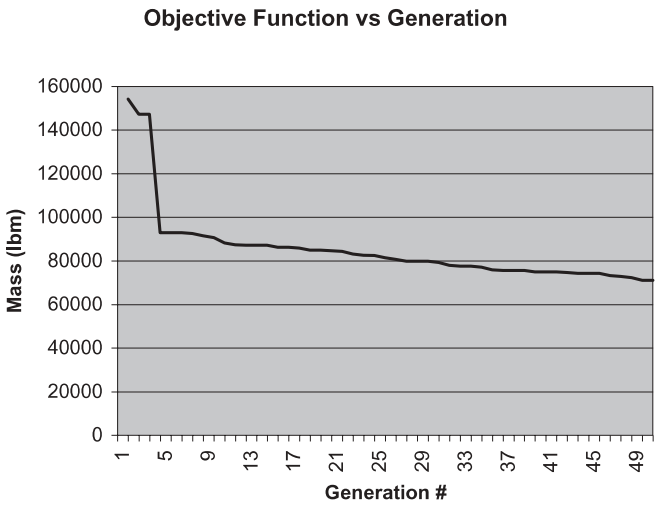

Fig. 4 Design history showing objective function versus generation (TRUSS2)

Table 4 Results of TRUSS2 problem

(a) High-cost cluster

Number of processes (machines)	LB version Time (s)	LB version Speedup	DLB version Time (s)	DLB version Speedup
1 (1)	6762	1.0	6762	1.0
3 (2)	3366	2.0	3370	2.0
4 (3)	2269	2.98	2260	2.99
5 (4)	1709	3.96	1701	3.98
6 (5)	1382	4.89	1376	4.91
7 (6)	1155	5.85	1148	5.89
11 (5)	716	9.44	690	9.8
15 (7)	524	12.9	510	13.25

(b) Low-cost cluster

Number of processes (machines)	LB version Time (s) C1/C2	LB version Speedup	DLB version Time (s) C1/C2	DLB version Speedup
1 (1)	18 491/11 930	1.0	18 491/11 930	1.0
3 (2)	7473	1.94	7301	1.99

shown in Table 4(a) and 4(b). The initial objective function value is 149 290 lbm and the final objective function value after 50 generations is 65 116 lbm.

Remark 2. Network traffic is much more in this example. With the LB approach, a total of 5 836 800 bytes are sent and received every generation. With the DLB approach, 48 000 point-to-point bytes are sent and received every generation, and 38 400 broadcast bytes are sent and received every generation. The amount of physical memory appears to be adequate for the problem being solved. When the system performance is monitored, the hard page faults are shown to be minimal.

Once again, the low-cost heterogeneous cluster shows an almost linear speedup. The DLB approach is more efficient, showing an ideal speedup because the communication traffic is much less compared with the LB approach. On the other hand, with the high-cost cluster, as observed in the cluster performance analysis, the DLB approach becomes more effective with increasing numbers of processes.

The design history is shown in Fig. 4.

5 Concluding remarks

The development and implementation of an MPI-enabled GA is discussed. Due to the very nature of genetic algorithms, linear speedup is possible with minimal effort – a good example of an embarrassingly parallel problem. The developed algorithms work well in both homogenous and heterogeneous network of workstations clusters. Such an environment can include shared as well as distributed memory. Due to the nature of the problems discussed in this paper and genetic algorithms, the heterogeneous cluster using COTS components performed as well as the more expensive homogenous cluster.

Computer systems and algorithms are evolving to a point where commodity computers/workstations are able to perform analyses that supercomputers were performing just a few years ago. For example, in an automotive-industry example, Sobeski *et al.* (2000) used MSC/NASTRAN for NVH analysis and sensitivity computations, and the crashworthiness analysis was performed by the RADIOSS CRASH code. By using 254 processors to solve the optimization problem, the solution as obtained in 1 day compared with an estimated 257 days if only a single processor was used.

There are other approaches that can be taken to improve the overall performance of the GA – better ac-

curacy and better efficiency. For example, the concept of using DGA with subpopulations and migration between these population islands is suitable for a parallel-computing environment (Fernandez *et al.* 2000). In addition, there is a wealth of research ideas and data available in the area of parallel genetic algorithms (Cantú-Paz 2000). These and other ideas are currently being investigated.

References

Chen, S.-Y.; Situ, J.; Mobasher, B.; Rajan, S.D. 1997: Use of Genetic Algorithms for the Automated Design of Residential Steel Roof Trusses. ASCE Press, 43–54

Chen, S.-Y.; Rajan, S.D. 1998: Improving the efficiency of genetic algorithms for frame designs. *Eng. Optim.* **30**, 281–307

Rajan, S.D.; Mobasher, B.; Chen, S.-Y.; Young, C. 1999: Cost-based design of residential steel roof systems: a case study. *Struct. Eng. Mech.* **8**(2), 165–180

Chen, S.-Y.; Rajan, S.D. 2000: A robust genetic algorithm for structural 0ptimization. *Struct. Eng. Mech.* **10**(4), 313–336

Biedron, R.T.; Mehrotra, P.; Nelson, M.L.; Preston, F.S.; Rehder, J.J.; Rogers, J.L.; Rudy, D.H.; Sobieski, J.; Storaasli, O.O. 1999: Compute as fast as the engineers can think! *NASA/TM-1999-209715*, September 1999

Eby, D.; Averill, R.C.; Gelfand, B.; Punch, III, W.F.; Mathews, O.; Goodman, E.D. 1997: An injection island GA for flywheel design optimization. In: *Proc. EUFIT '97 – 5th European Congress on Intelligent Techniques and Soft Computing*

Miki, M.; Hiroyasu, T.; Hatanaka, K. 1999: Parallel genetic algorithms with distributed-environment multiple population scheme. In: 3rd *WCSMO World Congress of Structural and Multidisciplinary Optimization*, Niagara Falls, NY

Chipperfield, A.; Fleming, P. 1996: Parallel genetic algorithms, chapter 39. In: Zomaya, A.Y. (ed.) *Parallel and Distributed Computing Handbook*, McGraw-Hill

Sarma, K.; Adeli, H. 2001: Bilevel parallel genetic algorithms for optimization of large steel structures. *Comput. Aid. Civil Infrastruct. Eng.* **16**, 295–304

Scott, S.D.; Samal, A.; Seth, S. 1995: HGA: a hardware-based genetic algorithm. *Proc. of the 1995 ACM/SIGDA Third International Symposium on Field-Programmable Gate Arrays*, 53–59, Monterey, CA

MPI Software Technology, MPI-Pro Version 1.6.3, 2002

Sobieski, J.S.; Kodiyalam, S.; Yang, R.J. 2000: Optimization of car body under constraints of noise, vibration, and harshness (NVH), and crash. *Proc. 41st AIAA Structures, Structural Dynamics and Materials (SDM)*, Atlanta, GA

Fernandez, F.; Tomassini, M.; Punch, W.; Sanchez, J.M. 2000: Experimental study of multipopulation parallel genetic programming. In: *Genetic Programming, Proc. of EuroGP2000, Springer, Lecture Notes in Computer Science* **1802**, 283–293

Cantú-Paz, E. 2000: *Efficient and Accurate Parallel Genetic Algorithms*. Boston, MA: Kluwer