Advances in Engineering Software 40 (2009) 975–985

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Synchronous parallelization of Particle Swarm Optimization with digital pheromones

Vijay Kalivarapu *, Jung-Leng Foo, Eliot Winer

Mechanical Engineering Department, Virtual Reality Applications Center, 1620 Howe Hall, Iowa State University, Ames, IA 50011, USA

ARTICLE INFO

Article history:
Received 9 January 2008
Received in revised form 9 March 2009
Accepted 6 April 2009

Keywords:
Particle Swarm Optimization
Digital pheromones
Synchronous parallelization
Multi-modal design spaces

ABSTRACT

In this paper, Particle Swarm Optimization (PSO) using digital pheromones to coordinate swarms within *n*-dimensional design spaces in a parallel computing environment is presented. Digital pheromones are models simulating real pheromones emitted by insects for communication to indicate suitable food or nesting location. Particle swarms search the design space with digital pheromones aiding communication within the swarm during an iteration to improve search efficiency. Previous work by the authors demonstrated the capability of digital pheromones within PSO for searching the global optimum with improved accuracy, efficiency and reliability in a single processor computing environment. When multiple swarms explore and exploit the design space in a parallel computing environment, the solution characteristics can be further improved. This premise is investigated through deploying swarms on multiple processors in a distributed memory parallel computing environment. The primary hurdle for the developed algorithm was bandwidth latency due to synchronization across processors, causing the solution duration due to each swarm to be only as fast as the slowest participating processor. However, it has been observed that the speedup and parallel efficiency improved substantially as the dimensionality of the problems increased. The development of the method along with results from six test problems is presented.

1. Introduction

Heuristic evolutionary search methods such as Genetic Algorithms (GAs) and Simulated Annealing (SA) are capable of exhaustively exploring *n*-dimensional design spaces to find optimal solutions. Their probabilistic nature provides distinct advantages over deterministic methods in finding global optimums, especially in multi-modal optimization problems. These methods are iterative in nature and traditionally do not require derivative calculations, thus allowing discontinuous design spaces to be explored. Although heuristic methods cannot mathematically guarantee decreasing an objective function or locating an optimum, they are often more reliable than deterministic methods such as a conjugate direction search. However, a downside to heuristic methods is their computational expense and sometimes implementation complexity. The recent advancements in processor and communication network technologies have fortunately catered to the growing computational demands through increased availability of low cost computing power and system memory. Multi-processor communication tools such as MPI [1], and PVM [2] are readily available for use on these hardware for massive parallel computing.

PSO [3,4] is a zero-order, population based heuristic method retaining many characteristics of evolutionary search algorithms. Compared to GAs and SA, it is simpler to implement and there are a smaller number of parameters to adjust [5,6]. PSO has been recently added to the list of global search methods [7,8] due to its reliability in finding the global optimum for a wide range of problems. In PSO, a randomly generated particle swarm (a collection of particles) propagates towards a global optimum in a series of iterations, based on the information provided by two members – the best position of a swarm member in its history trail (*pBest*), and the best position attained by all swarm members (*gBest*). Based on this information, a basic PSO generates a velocity vector indicating the direction of the swarm movement, and updates the location of the particles. The first drawback of this method is that the particle updates are influenced by a limited number of factors. At any instance, each swarm member is directed only by two candidates – *pBest* and *gBest*. Having just these two candidates potentially impedes the desirable exploratory characteristics. In an *n*-dimensional design space, information from these two candidates alone will not always suffice to propagate the swarm towards the global optimum efficiently. A second drawback is that the method is initial condition dependent. Poor locations specified by *pBest* and *gBest* in the initial stages of optimization can potentially offset the swarm from efficiently or accurately attaining the neighborhood of the solution in the design space [9]. This results in the

* Corresponding author. Tel.: +1 515 294 3092; fax: +1 515 294 5530.
E-mail addresses: vkk2@iastate.edu, vijaykiran@gmail.com (V. Kalivarapu).

0965-9978/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2009.04.002

swarm either being trapped in a local minimum or taking excessive amount of time to converge on the global optimum.

Previous work by the authors [10] addressed the above drawbacks through implementing digital pheromones in PSO. The work demonstrated substantial performance improvements of PSO in terms of efficiency, accuracy and reliability in locating the global optimum in multi-modal design spaces on a single processor workstation. However, when multiple swarms are deployed in an n-dimensional design space, it is theorized that the search efficiency can further be improved. This can be achieved through leveraging parallel computing environments where multiple swarms search the design space as a function of the available number of processors. A method to test this theory has been implemented into software and tested on a number of multi-modal benchmarking test cases. Background on the method, the method development, and results are all presented in this paper.

2. Background

2.1. Parallelization

The primary requirement for parallelization is the ability of the method to decompose into segments for multi-processor operation. In addition, the two highly desirable characteristics for parallelization are: (a) scalability – the ability to adapt to any number of processors with no/minimal changes and (b) processor load balancing – use of the available number of processors to the full extent without any processor substantially running idle. Population based optimization methods such as GA and PSO are a natural fit for parallelization because the method parameters do not limit the number of processors that can be used for solving the problem.

Parallelization can be synchronous or asynchronous. Synchronous parallelization facilitates a step wise parallel execution of tasks. Coarse decomposition schemes are examples of synchronous parallelization where, for instance, each processor has its own swarm exploring the design space. Solutions obtained from different processors are synchronized and gathered on a common processor (usually, the root processor) to evaluate the final global optimum. The synchronization is achieved through the use of a barrier function in Message Passing Interface (MPI), the most commonly used interface for parallel programming. Asynchronous parallelization is the dividing of a sequential algorithm into autonomous tasks each of which can be carried out on different processors. Dependencies among the tasks are modeled by message passing or through shared memory [11], depending upon the hardware configuration.

2.2. Particle Swarm Optimization

PSO is a population based zero-order optimization method that exhibits several evolutionary characteristics similar to GAs and SA. These are: (1) initialization with a population of random solutions, (2) design space search for an optimum through updating generations of design points, and (3) update based on previous generations [12].

PSO is based on a simplified model of the social behavior exhibited by the swarming behavior of insects, birds, and fish. In this analogy, a swarm member (particle) uses information from its past behavior (best previous location – *pBest*) and the behavior of the rest of the swarm (the overall best particle – *gBest*) to determine suitable food or nesting locations (local and global optimums). The algorithm iteratively updates the search direction of the swarm propagating towards the optimum. Eqs. (1) and (2) define the mathematical simulation of this behavior:

$$V_{iter+1,i}[] = w_{iter} \times V_{iter,i}[] + c_1 \times rand_{p,i}^{iter+1}() \times (pBest_i[] - X_i[]) + c_2 \times rand_{g,i}^{iter+1}() \times (gBest_i[] - X_i[]) \quad (1)$$

$$X_{iter+1,i}[] = X_{iter,i}[] + V_{iter+1,i}[] \quad (2)$$

$$w_{iter+1} = w_{iter} \times \lambda_w \quad (3)$$

Eq. (1) represents the velocity vector update, representing the direction and magnitude of ith swarm member in a basic PSO in iteration '*iter*'. Each successive iteration is represented by '*iter* + 1'. The square braces in Eqs. (1) and (2) indicate an array meaning that the corresponding value (e.g., *pBest*) is computed for each design variable. $rand_{p,i}^{iter+1}()$ and $rand_{g,i}^{iter+1}()$ are unique random numbers generated between 0 and 1 for *pBest* and *gBest* components separately for each swarm member in each iteration. The parentheses in $rand_p(\,)$ and $rand_g(\,)$ indicate that they are random number generating functions within the computer code. This ensures swarm diversity, meaning that the search is not linear in an n-dimensional space. More about swarm diversity can be obtained from [13]. c_1 and c_2 are user definable confidence parameters. Typically, these are set to values of 2.0. '*pBest*' represents the best position of the particle in its history trail, and '*gBest*' represents the best particle location in the entire swarm. w_{iter} is termed "inertia" weight, and is used to control the impact of a particle's previous velocity on the calculation of the current velocity vector. A large value for w_{iter} facilitates global exploration, which is particularly useful in the initial stages of an optimization. A small value allows for more localized searching, which is useful as the swarm moves toward the neighborhood of the optimum [14,15]. These characteristics are attributed to the swarm by implementing a decay factor, λ_w for the inertia weight, as shown in Eq. (3). Eq. (2) denotes the updated swarm location in the design space.

Ever since the inception of PSO in 1995, a significant number of modifications have been made to the basic algorithm for realizing performance improvements. Natsuki and Iba [16], and Hu et al. [17] have explored the possibilities of performance improvement through introducing mutation factors in PSO, similar to the ones used in GAs. Various methods for constraint handling using PSO have been addressed. Venter and Sobieszczanski-Sobieski [18] implemented a quadratic exterior penalty function method to solve non-linear constrained optimization problems. Hu and Eberhart [19] modified the basic PSO method so that the swarm is repeatedly initialized until all constraints are satisfied while also forcing *pBest* and *gBest* to be feasible in every iteration. Sedlaczek and Eberhard [20] implemented the Augmented Lagrangian Method for solving constrained non-linear optimization problems. Ray and Saini [21] developed a method to improve swarm movement within the design space through information sharing between individual particle members. They have successfully implemented this strategy in solving both unconstrained and constrained problems as well. A preliminary implementation of digital pheromones for use by PSO has been addressed by the authors to improve design space exploration in single and parallel computing environments [22,23]. Other parallel implementations of PSO include a coarse grain synchronous parallelization scheme by Schutte et al. [24], where fitness evaluations are concurrently performed on different processors and the particle positions are updated after synchronizing results from the participating processors at the end of each iteration. Koh et al. [25] and Venter and Sobieski [26] implemented an asynchronous parallelization of PSO where the necessity for a barrier synchronization at the end of each iteration is eliminated. This approach improved the parallel speedup and efficiency while also maintaining load balance between processors.

PSO was used and modified for solving multi-objective problems as well [27,28]. Gao et al. have obtained improvements in PSO, through the use of a virus operator that propagates partial genetic information in the swarm by infection operators for enhanced de-

sign space search [29]. Some of the recent advancements include solving traveling salesman problems using discrete PSO methods [30–33]. Penalty function approaches have been used to solve mixed discrete non-linear problems using PSO [34]. Discrete PSO methods have been known to solve constrained optimization problems as well, and Yang et al. have demonstrated it through converting constraint satisfaction problems into discrete optimization problems [35]. Other areas include developments in the areas of solving integer programming [36] and continuous variable problems [37]. Parsopoulos and Vrahatis have demonstrated the use of PSO for solving a wide range of problems including multi-objective, minimax, integer programming problems [38]. The same authors have developed methods to compute all global minimizers of an objective function using PSO [39]. Similarly, He et al. presented methods to solve various mechanical design problems that tackled mixed variable types – integer, discrete, and continuous variables [40]. A 'flyback' constraint handling mechanism was also introduced in this research to maintain a feasible population. A substantial amount of success has been achieved in utilizing PSO for applications such as aircraft design [41,42], topology and shape optimization [5,43], structural optimization [44,45], wireless network routing problems [46], optimization in manufacturing and production operations [47,48], collision detection problems [49], and detection of optimal paths for unmanned aerial vehicles (UAVs) [50,51] to name a few.

2.3. Digital pheromones

Pheromones are chemical scents produced by insects essentially as a means of communication in finding suitable food and nesting locations. The more insects that travel a path, the stronger the pheromone trail. A digital pheromone works on the same principle and is analogous to a natural pheromone in that it is a marker to determine whether or not a region in the design space is promising for further investigation. Digital pheromones have been used in applications such as the automatic adaptive swarm management of unmanned aerial vehicles (UAVs) [52,53]. In this research, the implementation of digital pheromones allowed simulated swarms of UAVs to automatically adapt and navigate in potentially hazardous environments, dramatically reducing the requirement of human operators at the ground control stations. Other applications of digital pheromones include ant colony optimization for solving minimum cost paths in graphs [54–56], and solving network communication problems [57].

2.4. PSO and digital pheromones

The concept of digital pheromones is relatively new [58], and has not been explored to its full potential for investigating n-dimensional design spaces. The benefits of digital pheromones from swarm intelligence and the adaptive applications described above can be merged into PSO to improve design space exploration, particularly for a multi-modal optimization problem where swarm communication is essential to locating the global optimum. In a basic PSO algorithm, the swarm movement is governed by the velocity vector computed in Eq. (1). Each swarm member uses information from its previous best and the best member in the entire swarm at any iteration. However, multiple pheromones released by the swarm members could provide more information on promising locations within the design space when the information obtained from *pBest* and *gBest* are insufficient or inefficient. Fig. 1 displays a scenario of a swarm member's movement whose direction is guided by *pBest* and *gBest* alone.

If $c_1 \gg c_2$, (coefficients appearing in Eq. (1)) the particle is attracted primarily towards its personal best position. On the other hand, if $c_2 \gg c_1$, the particle is strongly attracted to the *gBest* position. In the scenario presented in Fig. 1, a bias towards neither

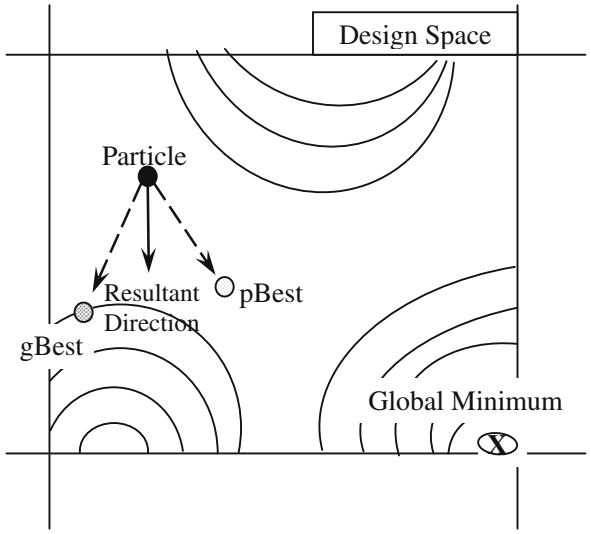

Fig. 1. Particle movements in a basic PSO.

pBest nor *gBest* leads the swarm member to the global optimum. If the swarm member does locate the global optimum, it will take additional computation that could have been avoided if communication amongst swarm member during an iteration was occurring. Fig. 2 shows the effect of implementing digital pheromones into the velocity vector. An additional pheromone component causes the swarm member to result in a direction different from the combined influence of *pBest* and *gBest* thereby increasing the probability of finding the global optimum.

The research presented in this paper develops and implements this idea by exploring the benefits of an additional component into the velocity vector for achieving improved solution characteristics in PSO, in a parallel computing architecture. The remaining sections of this paper focus on the method development, implementation into software, and then evaluation through several test cases.

3. Methodology

3.1. Overview of PSO with digital pheromones

Fig. 3 is an overview of PSO with steps involving digital pheromones highlighted. The method initialization is similar to a basic

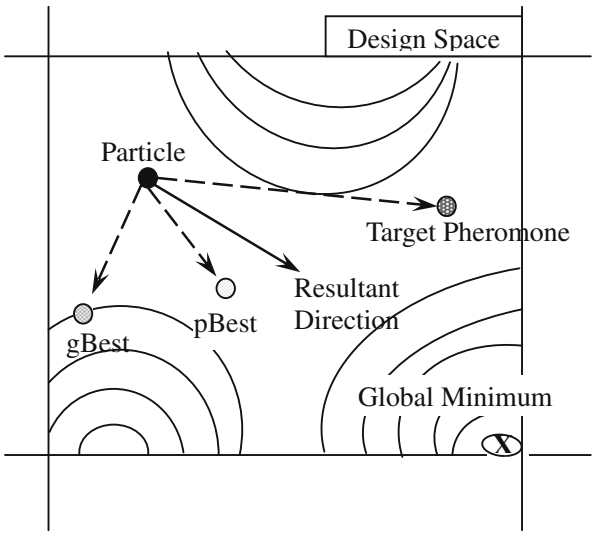

Fig. 2. Particle movement with digital pheromone.

PSO except that 50% of the swarm within the design space is randomly selected to release pheromones in the first iteration. This parameter is user-defined, but experimentation has shown 50% to be a good default value. For subsequent iterations, each swarm member that realizes any improvement in the actual objective function value is allowed to release a pheromone. Pheromones from the current as well as the past iterations that are close to each other in terms of the design variable value are merged into a new pheromone location. Therefore, a pheromone pattern across the design space is created, while keeping the number of pheromones manageable. In addition, the digital pheromones are decayed every iteration just as natural pheromones. Based on the current pheromone level and its position relative to a particle, a ranking process is used to select a target pheromone for each particle in the swarm. This target position towards which a particle will be attracted is added as an additional velocity vector component to *pBest* and *gBest*. This procedure is continued until a prescribed convergence criterion is satisfied.

3.2. Merging of digital pheromones

In order to populate the design space with an initial set of digital pheromones, 50% of the population is randomly selected to release pheromones, regardless of the objective function value. This is done so as to ensure a good spread of the swarm for exploring the design space in the initial stages of the optimization process. For subsequent iterations, the objective function value for each particle in the population is evaluated and only particles finding an improvement in the actual objective function value will release a pheromone. Any newly released pheromone is assigned a level *P*, with a value of 1.0. The pheromone levels are normalized between 0.0 and 1.0. Just as natural pheromones produced by insects decay in time, a user defined decay rate, λ_P (defaulting to 0.95), is assigned to the pheromones released by the particle swarm. Digital pheromones are decayed as the iterations progress forward to allow a swarm member to propagate toward a better design point by increasing the chances of attraction to a newer pheromone location with a better objective function value.

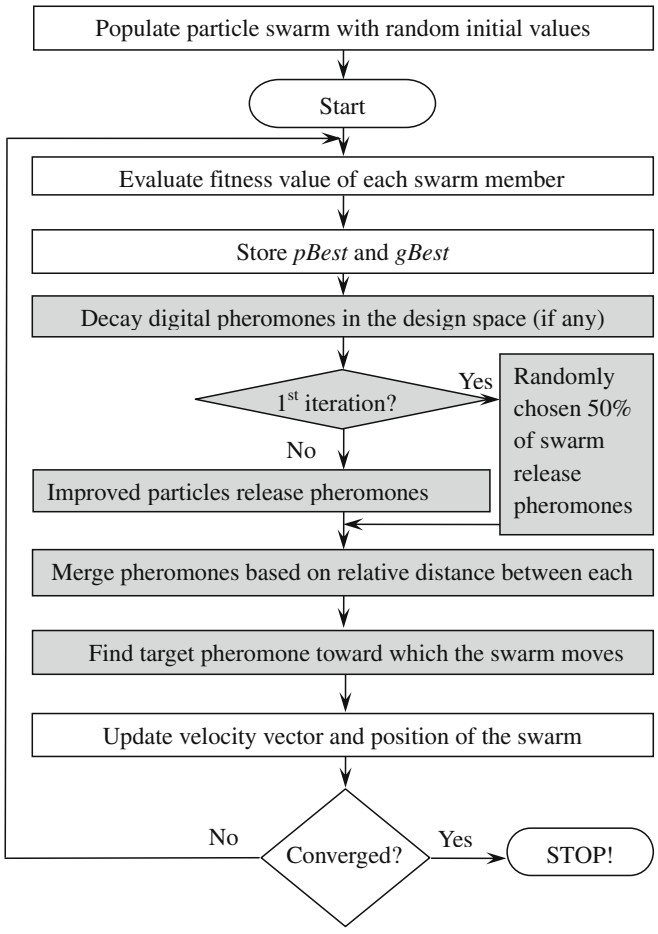

Fig. 3. Flowchart of PSO with digital pheromones.

As iterations progress, the number of pheromones in the design space can become very large. To address this issue, pheromones that are closely packed within a small region of the design space are merged together. To check for merging, each pheromone is assigned a 'Radius of Influence' (*ROI*). The value of this *ROI* is a function of the pheromone level and the bounds of the design variables. Any two pheromones for a design variable less than the sum of the ROIs are merged into one. A resultant pheromone level, whose location is the mid-point of the two merged pheromones, is then computed for the merged pheromones. Through this approach, regions of the design space with stronger resultant pheromone levels will attract more particles and therefore, pheromones that are closely packed would indicate a high chance of optimality.

Also similar to the pheromone level decay, the *ROI* also has its own decay factor, λ_{ROI}, whose value is set equal to λ_P as a default. This is to ensure that both the pheromone levels and the radius of influence decay at the same rate.

3.3. Proximity analysis to determine target pheromone

With numerous digital pheromones generated within the design space, a target pheromone needs to be identified for each swarm member. This is based on: (a) the distance between a particle and pheromone, and (b) the pheromone level. For each particle, a target pheromone attraction factor *P*′ is computed to this effect, which is a product of the pheromone level and the normalized distance between the particle and the pheromone. Eq. (4) shows how the attraction factor *P*′ is computed, and Eq. (5) computes the distance between the pheromone and each particle in the swarm. The variable $range_k$ is the difference in the upper and lower limits of *k*th design variable:

$$P' = (1-d)P \tag{4}$$

$$d = \sqrt{\sum_{1}^{k}\left(\frac{Xp_k - X_k}{range_k}\right)^2}$$
$$k = 1:n \ \#\ \text{of design variables}$$
$$Xp = \text{location of pheromone} \tag{5}$$
$$X = \text{location of particle}$$

3.4. Velocity vector update

The velocity vector update, shown in Eq. (6) implements digital pheromones described in the methodology Sections 3.1 and 3.2 as a new component, called the target pheromone component:

$$\begin{aligned} V_{iter+1}[] = {} & w_{iter} \times V_{iter,i}[] \\ & + c_1 \times rand_{p,i}^{iter+1}() \times (pBest_i[] - X_i[]) \\ & + c_2 \times rand_{g,i}^{iter+1}() \times (gBest_i[] - X_i[]) \\ & + c_3 \times rand_{T,i}^{iter+1} \times (TargetPheromone_i[] - X_i[]) \end{aligned} \tag{6}$$

$rand_{p,i}^{iter+1}()$, $rand_{g,i}^{iter+1}()$, and $rand_{T,i}^{iter+1}()$ are unique random numbers generated between 0 and 1 for *pBest*, *gBest* and target pheromone components separately for each swarm member in each iteration. The parentheses in $rand_p(\ )$, $rand_g(\ )$, and $rand_T(\ )$ indicate that they are random number generating functions within the

computer code. These values are updated in each iteration within the velocity vector equation, resulting in improved swarm diversity. The square braces in Eq. (6) indicate an array meaning that the corresponding value (e.g., *TargetPheromone*) is computed for all design variables.

c_3 is a user defined confidence parameter for the pheromone component of the velocity vector similar to c_1 and c_2 in a basic PSO. c_3 combines the knowledge from the cognitive and social components of the velocity of a particle, and complements their deficiencies. In a basic PSO, the particle swarm does not have a memory of the entire path traversed in the design space apart from the best position of an individual particle (*pBest*) and the best member's position in the entire swarm (*gBest*). The target pheromone component addresses this issue. It is a container that functionally stores the trail path of the swarm and utilizes the best features of *pBest* and *gBest* in steering towards a promising location in the design space. The confidence parameter c_3 determines the extent of influence a target pheromone can have on the swarm when the information from *pBest* and *gBest* alone are not sufficient or efficient to determine a particle's next move. The use of the target pheromone relies heavily on *pBest* and *gBest*. If $c_3 = 0$, there is no influence of pheromones and the swarm behaves as if in a regular PSO. If either of c_1 or c_2 is 0 and $c_3 > 0$, then the target pheromone location is essentially determined only by the non-zero component of *pBest* or *gBest* and propagated into the velocity vector. This creates a bias thereby doubling the influence of non-zero *pBest* or *gBest* components on the swarm. This means that the swarm either explores or exploits the design space with double the intensity, either of which will prevent the swarm from converging. It is therefore essential that the influence of *pBest* and *gBest* be balanced (i.e. equal) for the pheromone component to provide accurate assistance in reaching the optimum. The addition of this pheromone component to PSO increases the swarm's diversity, resulting in an improved search in the design space. However, improvement in diversity most certainly lacks frame invariance [59]. This paper focuses on increasing the diversity in the swarm using digital pheromones than frame invariance.

Although analytical determination of a value for c_3 is out of the scope of this research, an empirical value has been determined through experimentation. A value between 2.0 and 5.0 has shown good performance characteristics and solved a variety of problems. An inertia weight, w_i of value 1.0 is initially chosen to preserve the influence of the velocity vector from previous iterations, and gradually decreased using an inertia weight decay factor similar to the one used in a basic PSO.

3.5. Move limits

The additional pheromone term in the velocity vector update can considerably increase the computed velocity. Therefore, a move limit was applied to impose an upper bound on the maximum value of the velocity vector. To ensure a fair amount of freedom in exploring the design space, the swarm is allowed to digress up to 10% of the range of the design variables initially. A decay factor of 0.95 is applied to this move limit in subsequent iterations. This means that the freedom to explore the design space decreases as the iterations progress forward. The initial 10% value in the move limit showed good performance characteristics upon in the test cases used.

3.6. Coarse grain parallelization

3.6.1. Rationale for parallelization

The two major drawbacks of PSO are: (a) the direction of a swarm member is influenced by insufficient number of factors – *pBest* and *gBest*, and (b) a poor location specified by them in the initial stages potentially offsets the swarm in reaching the global optimum efficiently. It can be speculated that with a higher swarm size, the chances of locating the optimum increases. On the contrary, the swarm activity due to increased swarm size causes significant movement when it approaches the optimum and may not converge. A potential solution to this problem, which is also the premise of the research presented in this paper, is to have multiple swarms searching the design space for the optimum. Multiple independent swarms traversing the design space independently can garner significantly more information on the design space than a single swarm with a larger population size with increased efficiency. The question now remains how this methodology is implemented.

An equivalence of '*n*' swarms can be achieved on a computer workstation by deploying '*n*' swarms one after another as each converges. However, the efficiency of the method is greatly decreased because it potentially takes at least '*n*' times the number of seconds for all swarms to report results. Another approach would be to deploy '*n*' independent swarms simultaneously on a computer workstation through a threaded code. In this approach, a processor can spawn multiple processes, each handling an independent swarm. However, the processor load increases substantially thereby resulting in degraded performance and increased time to solve.

Given these circumstances, a logical solution would be is to parallelize the computations using MPI or PVM communication layers. These APIs are simple to implement and effectively distribute information between processors thereby easing the computational intensity on a single processor and yet are able to deploy multiple independent swarms. Where a serial code takes '*np*' seconds to solve a problem using '*p*' swarms, an ideally formulated parallel code would only take '*n*' seconds when '*p*' swarms are delegated each to a processor, dramatically reducing the solution time. However, parallelization comes at a considerable expense – network latencies. Communication between processors is currently limited by the available network technologies and every instance of a data transfer between processors is as fast as the slowest network connection. Therefore, benefits can be reaped only when the communication between the processors is kept to a minimum.

3.6.2. Coarse grain parallel implementation

A coarse grain parallelization scheme has thus been devised that address the above issues and avoids the pitfalls of computational intensity, solution efficiency, and network latencies. Fig. 4 shows the schematic of the implementation.

As seen in Fig. 4, each processor has a randomly initialized swarm propagating on its own. In this approach, each swarm self propagates towards the optimum with no communication between other swarms. Although it is logical to think that their communication can provide a better investigation of the design space, the network latency costs for message passing between processors can dominate the benefits of self driven swarm efficiency. Since communication between processors is kept to a minimum in the proposed approach, network latency issues do not interfere with the solution efficiency until barrier synchronization takes place. Upon reaching the optimum on all processors, the results are gathered on a single (root) processor through a one-time communication signal and sorted for the best objective function. The results are then reported. Since each swarm independently operates on a processor, the overall solution time is as fast as the slowest processor. It can also be seen in this approach that the scalability of the method to the available number of processors is inherently addressed. The number of swarms deployed can therefore be as many as the number of available processors.

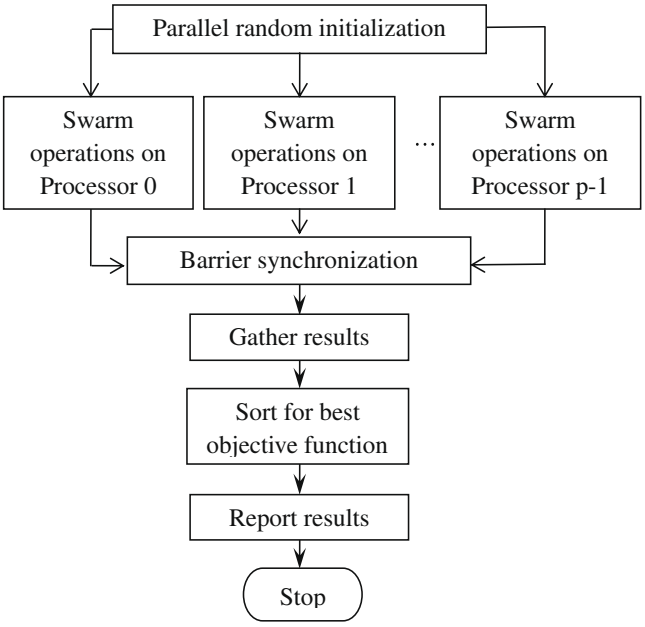

Fig. 4. Schematic of synchronous coarse grain parallelization.

3.6.3. Parallel performance evaluation

The output from parallelizing an algorithm is typically measured and compared in terms of speedup and parallel efficiency. The speedup defines how fast a code runs in parallel; it is a ratio of the amount of time the code spends in communication to the amount of time it spends on computing. If the time taken to run a code on one processor is t_1 seconds, and the time it takes to run the same code on 'p' processors is t_p seconds, then the parallel speedup is given by Eq. (7):

$$\text{Speedup} = \frac{t_1}{t_p} \tag{7}$$

Parallel efficiency is a percentage measure of how well the available processors are used. In other words, it provides information on how well the load balancing is maintained. Eq. 8 shows the procedure for calculating parallel efficiency:

$$\text{Parallel Efficiency} = \frac{\text{Speedup}}{p} \tag{8}$$

4. Results

4.1. Overview

The results section is divided into two main categories:

(a) Performance evaluation of PSO with and without digital pheromones: accuracy, efficiency, and reliability of PSO with digital pheromones are compared against basic PSO in a parallel computing architecture.
(b) Parallel performance: evaluation of the developed method for adaptability to the parallel computing architecture. This involves evaluating parallel speedup and efficiencies of PSO with digital pheromones.

Six unconstrained minimization problems of varying dimensionality were used as test cases to study the developed method as shown in Table 1. The published solution for these problems are 0.0000 and their full mathematical descriptions for these problems can be found in [60–62].

In addition to the parameters of a basic PSO, the implementation of digital pheromones in PSO required three additional parameters namely c_3, pheromone decay rate and move limit decay rate as explained in Section 3. In the previous work by the authors, default values have been established for these parameters through a thorough testing of the method with 128 different pheromone parameter settings on problems ranging from two dimensions through 50 dimensions. Parallelization was implemented on the test cases with all 128 different pheromone parameter settings on problems ranging from two dimensions through 50 dimensions. The findings for the pheromone parameter values that best addressed the test problems were consistent with the values established by the authors in their previous work [10], and hence have been used as default values for reporting in this paper. Though customization of parameters could potentially improve the solution characteristics, the following parameter values catered well for most problems:

Table 1
Test problem matrix.

Problem number	Problem	Dimensionality
4.1	Dixon and Price function	15
4.2	Ackley's path function	20
4.3	Levy function	25
4.4	Sum of squares function	30
4.5	Spherical function	40
4.6	Griewank function	50

- c_3 = 5.0 with no decay,
- pheromone decay = 0.95, and
- move limit decay = 0.95.

The swarm size was defined as 10 times the number of design variables, and was limited to a maximum of 500 as the dimensionality of the problems increased. A total of 20 trial runs were performed for each test case, with and without digital pheromones. All test cases were solved on 2, 4, and 8 Intel Xeon processors (3.06 GHz) of a RedHat Linux cluster that houses 2 GB system memory per node and high bandwidth Myrinet network switches. The algorithm was implemented using the C++ programming language and MPI communication libraries (MPICH implementation) were used for data distribution between processors.

4.2. Performance evaluation of PSO with and without digital pheromones

Table 2 provides a summary of objective function values obtained from the test runs on 2, 4, and 8 processors. Upon convergence on all participating processors, the root processor gathered the solution information and sorted for best objective function values. The root processor retained these values and the others were discarded. Results displayed in Table 2 indicated these sorted values. The table contains three markers that assess the performance of the developed method – average objective function value, smallest objective function value, and the standard deviation. The smallest objective function is the lowest value obtained in 20 trial runs for each test problem. The results in the table show that PSO with digital pheromones (designated with a P) consistently displayed superior performance when compared with solutions from a basic PSO (designated with a B).

Since the published solutions for the problems in Table 1 are 0.000, there was no measure to determine the percentage accuracy. Therefore, a tolerance was given and accuracy was measured based on the number of times the obtained solution was within the tolerance limits. For example, a tolerance limit of ±0.5 was assigned for a 20 design variable problem. If the solution was within this tolerance limit 85 times in 100 runs of the problem, the solution accuracy was 85%.

Table 2
Summary of solutions from solving problems 4.1–4.6.

	Objective function (2 processors)			Objective function (4 processors)			Objective function (8 processors)		
	Average	Smallest	Std. dev.	Average	Smallest	Std. dev.	Average	Smallest	Std. dev.
4.1 (B)	14.170	0.002	35.015	34.822	0.003	126.622	3.628	0.001	13.676
4.1 (P)	0.211	0.001	0.643	0.098	0.001	0.324	0.441	0.001	1.083
4.2 (B)	5.456	2.245	3.504	6.181	1.991	4.031	5.946	1.900	4.013
4.2 (P)	0.354	0.002	0.723	0.393	0.004	0.890	0.133	0.002	0.399
4.3 (B)	0.134	0.131	0.002	0.133	0.130	0.002	0.134	0.131	0.003
4.3 (P)	0.131	0.130	0.001	0.131	0.130	0.001	0.131	0.130	0.001
4.4 (B)	3.166	0.259	3.212	1.531	0.273	1.070	2.178	0.220	2.313
4.4 (P)	0.228	0.006	0.304	0.236	0.005	0.220	0.174	0.003	0.278
4.5 (B)	0.035	0.017	0.014	0.036	0.012	0.018	0.031	0.009	0.019
4.5 (P)	0.002	0.001	0.001	0.002	0.001	0.001	0.002	0.001	0.001
4.6 (B)	1.151	1.067	0.064	1.146	1.051	0.044	1.155	1.061	0.074
4.6 (P)	0.012	0.003	0.009	0.011	0.004	0.006	0.009	0.003	0.005

Legend: 4.1 – Dixon and Price function 15D; 4.2 – Ackley's path function 20D; 4.3 – Levy function 25D; 4.4 – Sum of squares function 30D; 4.5 – Spherical function 40D; 4.6 – Griewank function 50D; (B) – results from basic PSO; (P) – results from PSO with digital pheromones implemented.

Fig. 5 shows the solution accuracy charts for the test problems across different number of processors. As evident from Fig. 5, the solution accuracy of PSO with digital pheromones was either equal or superior when compared to basic PSO. For example, in problem 4.2 (Ackley's 20D), the basic PSO was not able to solve the problem, whereas the pheromone PSO attained the solution within the specified tolerance limits 75 out of 100 runs on 2 and 4 processors. When tested with 8 processors, the accuracy of pheromone PSO increased to 90%.

The published solution for the 15-dimensional Dixon and Price function (problem 4.1) is 0.000 and the swarm in basic PSO was unable to locate the optimum on any of 2, 4 or 8 processor runs. The swarm reached closer to the optimum on the 8 processor cluster (3.628) but was still well out of the tolerance limits. On the other hand, PSO with digital pheromones solved the problem with a solution accuracy ranging between 80% and 95%. The standard deviation of pheromone PSO, as evident from Table 2, was substantially better than basic PSO as well. Also, in all the participating processors, the average solution time per run was at least 23% shorter for pheromone PSO than for basic PSO.

Both basic and pheromone PSO were able to solve the 25-dimensional Levy function (problem 4.3) within the tolerance limits as evident from the 100% solution accuracy in Fig. 5. Although the average, smallest and standard deviations between the methods (with and without digital pheromones) were quite close to each other on results from all processors, there was almost a 50% decrease in the solution time for pheromone PSO, showing that it was much more efficient than a basic PSO. Table 3 summarizes the average solution time and number of iterations for all test cases across 2, 4, and 8 processors.

The lowest objective function value obtained by the 30-dimensional sum of squares function (problem 4.4) by basic PSO was 0.220. Compared to the published solution of 0.000, this value is within the tolerance limits. However, the solution accuracy resulting from basic PSO method was only within the range of 5–25% when tested across 2, 4, and 8 processors. On the other hand, pheromone PSO was able to solve the problem within tolerance 80–95% of the time. Moreover, the solution time was 20% better than the basic PSO.

Both basic PSO and pheromone PSO solved the 40-dimensional spherical function (problem 4.5) within the tolerance limits resulting in 100% accuracy. That means that both basic PSO and pheromone PSO were able to find a solution within specified tolerance limits in the 20 trial runs. However, it can be observed from Table 2 that the average objective function evaluated by pheromone PSO

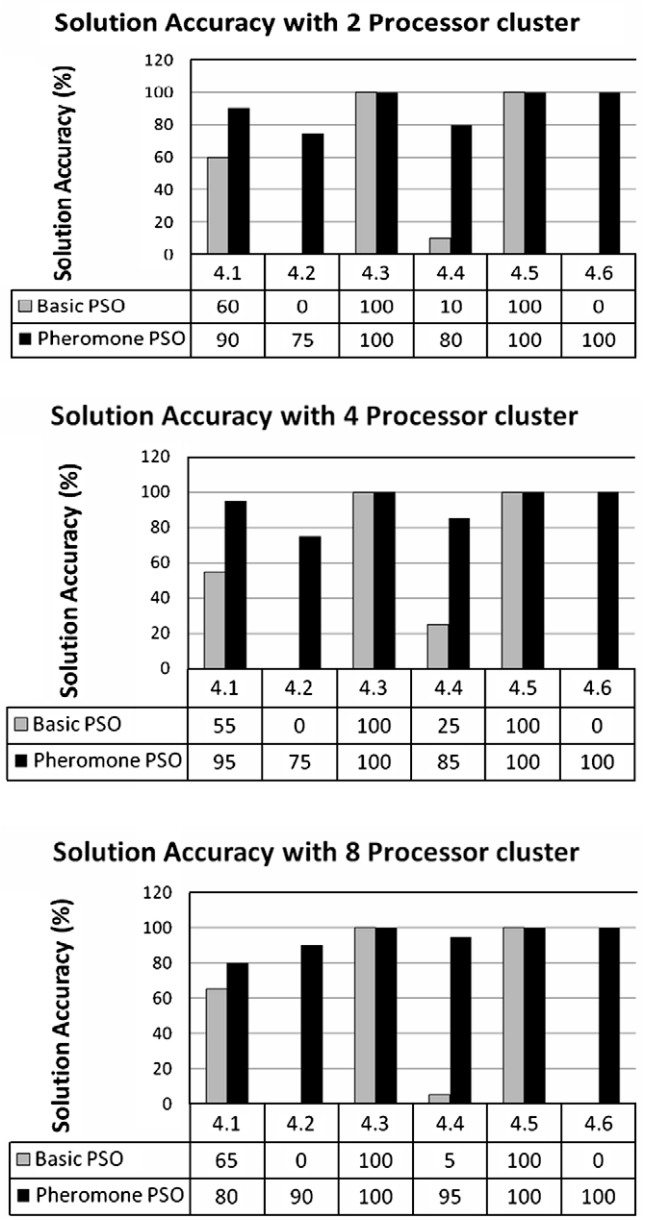

Fig. 5. Solution accuracy measure across 2, 4, and 8 processors.

Table 3
Summary of solution times and number of iterations from solving problems 4.1–4.6.

	2 Processors		4 Processors		8 Processors	
	Avg. # iterations	Avg. duration (s/run)	Avg. # iterations	Avg. duration (s/run)	Avg. # iterations	Avg. duration (s/run)
4.1 (B)	162.4	27.45	171.9	30.16	171.0	33.29
4.1 (P)	116.9	20.45	118.9	23.17	113.7	24.77
4.2 (B)	142.8	18.69	125.0	19.65	139.2	20.07
4.2 (P)	141.2	16.68	137.9	17.23	146.9	17.84
4.3 (B)	95.8	25.68	97.7	26.39	93.9	27.17
4.3 (P)	40.5	12.28	41.2	12.55	41.9	12.98
4.4 (B)	211.5	68.07	210.1	69.94	211.7	71.67
4.4 (P)	151.2	53.94	154.4	55.83	149.0	57.44
4.5 (B)	163.4	70.88	165.2	71.98	160.0	73.76
4.5 (P)	85.2	39.80	83.3	40.52	85.1	41.57
4.6 (B)	182.9	100.31	180.8	101.99	185.3	104.48
4.6 (P)	159.7	96.45	159.6	96.71	159.5	97.65

(0.002) is about 15 times better than the lowest average objective function returned by basic PSO (0.031). Moreover, the highest solution time for pheromone PSO (41.57 s/run on 8 processor cluster) is about 43% shorter when compared to that of basic PSO. Results from all the processors consistently showed a very small variation in the solution (standard deviation of 0.001), which proves the reliability of pheromone PSO.

A highly multi-modal 50-dimensional Griewank function (problem 4.6) was also attempted to solve using basic and pheromone PSO. While the basic PSO could not reach the global optimum on any of the 20 trials across 2, 4, and 8 processors, pheromone PSO was able to obtain a solution within the tolerance limits in all the trials, i.e., with a 100% accuracy. Moreover, the variation of the results as seen from the standard deviation values in Table 2, the pheromone PSO is significantly consistent in performance when compared to basic PSO. This also must be taken into account when considering iterations and solution times as basic PSO converged prematurely on every single solution run.

In all the test cases, the pheromone PSO displayed superior performance characteristics in terms of accuracy (closeness to published solution), efficiency (solution duration), and reliability (standard deviation) when implemented in a parallel architecture.

It was also observed that fine tuning of pheromone parameters can potentially improve the solution quality and duration. Since the digital pheromone parameters are user defined, they can be altered to suit to the type of problem being solved. For example, on the Ackley's 20D path function (problem 4.2), the reported average objective function value for 20 runs on an eight processor execution was 0.133 (Table 3) with $c_3 = 5.0$, pheromone decay = 0.95, and move limit decay = 0.95. However, when the pheromone parameters are fine tuned and moved to $c_3 = 5.0$, pheromone decay = 0.9, and move limit decay = 0.95 the average objective function value returned was 0.005. This is a 96% decrease in the objective function value with a 5% decrease in pheromone decay factor alone. On the other hand, the average objective function value changes to 1.192 when $c_3 = 5.0$, pheromone decay = 0.9, and move limit decay = 0.9. This is ~800% increase in the average objective function value when there is a 5% decrease in pheromone decay and move limit decay together. Although this is a sharp change in the objective function values with insignificant change in pheromone parameters, it suggests that the shape of the objective function could influence the outcome of the method. This means that although the default values assigned for pheromone parameters caters for most problems, improvement in solution quality through tweaking the parameters can be problem dependent.

4.3. Parallel performance

The parallel performance characteristics of pheromone PSO can be measured through speedup and efficiency calculations. Ideally, the parallel speedup should be equal to the number of processors, and the parallel efficiency should be 100%. However, due to communication latencies, these values do not usually reach these ideal values. Therefore, the measure of parallel performance is based on how close they are to ideal values. In this section, the parallel performance characteristics were evaluated and presented for pheromone PSO.

Fig. 6 shows the speedup characteristics of pheromone PSO plotted for all participating processors. The plot was generated based on the speedup values evaluated for each test problem running on 2, 4, and 8 processors. The *x*-axis portrays various test problems with their dimensionality and the *y*-axis shows the parallel speedup.

The plot shows that the parallel speedup was almost ideal when two processors were used alone. This means that the network latencies have a very negligible effect on the two swarms in a parallel architecture. When four swarms were deployed on a four processor system, the speedup did not reach the ideal as quickly as two processors. However, parallel speedup approached ideal value of 4.00 as the dimensionality of the problem increased to 40 (spherical function – problem 4.5). With eight swarms simultaneously deployed, a plateau was noticed at a non-ideal speedup (i.e., 7.00) when the problem dimensionality was between 20 (Ack-

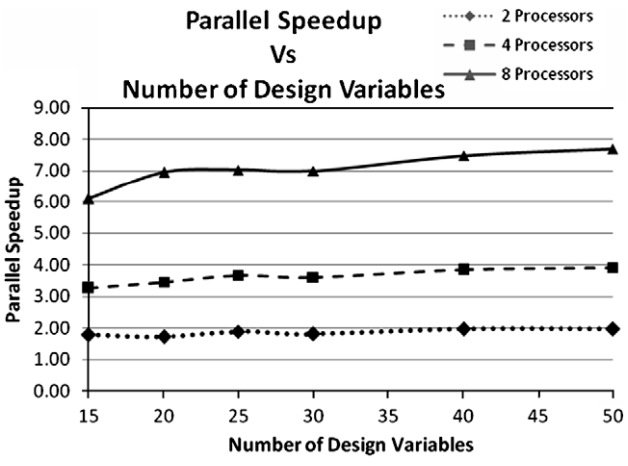

Fig. 6. Parallel speedup characteristics of pheromone PSO.

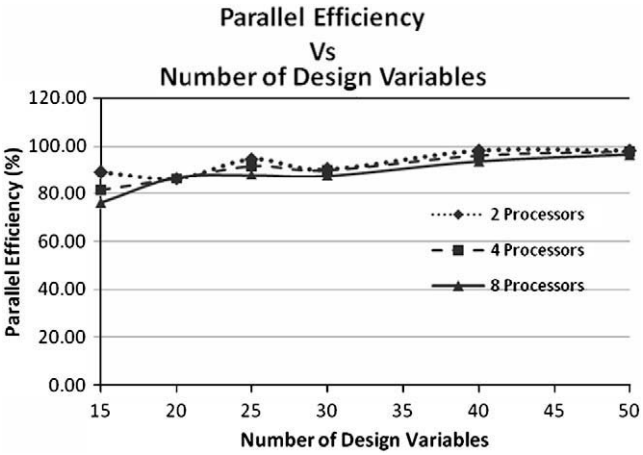

Fig. 7. Parallel efficiency characteristics of pheromone PSO.

ley's path – problem 4.2) and 30 (sum of squares – problem 4.4). The speedup gradually inched up towards ideal (i.e., 8.00) as the problem dimensionality increased to 50 (Griewank function – problem 4.6).

It can be inferred from these findings that latencies due to processor communication dominates the solution efficiency in lower dimensional problems. As the problem dimensionality increases along with the number of processors, the network latencies are offset and near ideal parallel speedups are attained.

Fig. 7 shows the parallel efficiency characteristics of pheromone PSO across the participating processors.

Parallel efficiency provides load balancing information and is dependent upon the speedup and the number of participating processors. A 100% parallel efficiency means that the system is perfectly load balanced. Fig. 7 shows that the parallel efficiency values ranged from 85% through 75% across 2, 4, and 8 processors for a 15-dimensional Dixon and Price function (problem 4.1). For this problem, this means that the load balancing worsened when the number of processors increased from 2 through 8. This trend more or less continued to the remaining problems as well. However, the decrease in parallel efficiency was negligible with higher dimensional problems (i.e., Griewank 50D). This means that the load balancing improved considerably as the dimensionality of the problems increased.

Fig. 8 shows the relation between parallel efficiency and the number of processors in the context of the six test problems. This figure is equivalent to Fig. 7, but the relation between parallel efficiency and number of processors in the context of test cases is better understood.

A pattern of increasing parallel efficiency can be observed in this figure. The 15-dimensional Dixon and Price function (problem 4.1) has the smallest parallel efficiency, while the 50-dimensional Griewank function (problem 4.6) shows traits of ideal parallelization characteristics. The parallel efficiencies for the test problems gradually increased as the dimensionality increased, which corroborates with the findings shown in Fig. 7 as well.

A similar speedup and efficiency study was also performed on the basic PSO method with multiple swarms traversing the design space, and the results concur with the pattern of findings reported above. Fig. 9 shows the plots from testing basic PSO.

The pheromone parameter settings used in reporting these results were empirically determined from previous work by the authors. However, with refined settings, the solution characteristics could be further improved. For example, solving the 40D sphere problem with the suggested pheromone parameters resulted in an average objective function value of 0.0019 that took an average of 41.57 s/run on an 8 processor computing environment. However, with tuned parameter settings, the best solution achieved was 0.0015 that took an average of 29.76 s/run. The altered parameter values showed a 20% improvement in the objective function value along with a 28% improvement in solution time. The changes in these parameters are problem dependent and currently do not have mathematical rules to ascertain the most optimal parameter settings. In spite of additional computations due to pheromone operations, solution times showed a trend for decreased solution times. This is attributed to the information provided by the digital pheromones thereby facilitating the swarms in propagating towards the global optimum faster.

5. Summary, conclusions, and future work

This paper explores the possibility of deploying multiple swarms in *n*-dimensional design spaces simultaneously in parallel computing environments. Six different benchmarking problems were tested to investigate the feasibility of this approach. Since this is a coarse grain parallelization, each participating processor shares the same code. Therefore, the solution accuracy or variation will not change significantly from a serial implementation of the code. Also, the results proved that the solution accuracy, efficiency, and reliability of the parallel implementation of pheromone PSO is significantly superior to the results from basic PSO.

The parallel performance studies show that almost ideal parallel speedups can be obtained, in spite of network latencies, as the

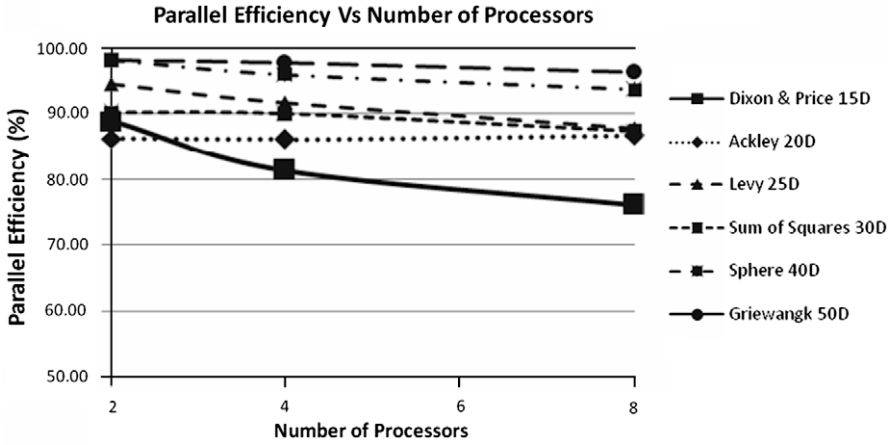

Fig. 8. Effect of number of processors on parallel efficiency.

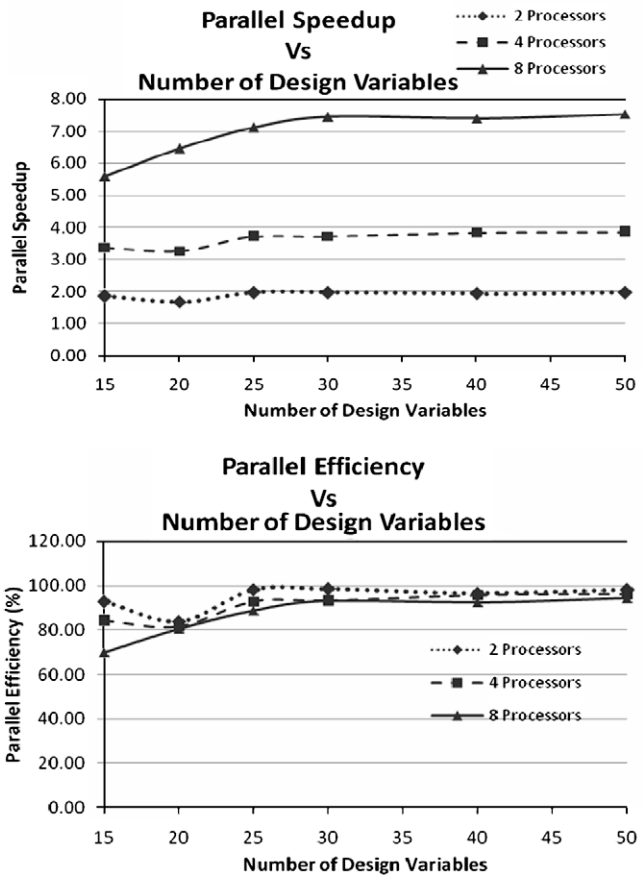

Fig. 9. (top) Speedup chart for basic PSO. (bottom) Parallel efficiency chart for basic PSO.

dimensionality and number of processors both increase. It was observed that lower dimensional problems are susceptible to lesser load balancing as the number of available processors increased. However, the load balance improved significantly as the dimensionality of the problems increased.

Refining the performance of digital pheromones to solve a wide range of optimization problems is an ongoing venture; one of the near future goals include testing and validating the robustness of the developed method on frame invariant rotational test problems, implementing asynchronous parallelization methods where the necessity for barrier synchronization can be eliminated altogether increasing the efficiency of the method. Other avenues of future work include developing mathematical rules for pheromone parameter settings, methods to solve constrained optimization problems, and multi-objective problems.

References

[1] Gropp W, Lusk E, Skjellum A. Using MPI – portable parallel programming with the message passing interface, 2nd ed. MIT Press; 1999. ISBN: 0262571323.
[2] Geist A, Beguelin A, Dongarra J, Manchek R, Jiang W, Sunderam V. PVM 3 user's guide and reference manual. Technical report ORNL/TM-12187, Oak Ridge National Laboratory, Knoxville, TN; 1994.
[3] Kennedy J, Eberhart RC. Particle Swarm Optimization. In: Proceedings of the 1995 IEEE international conference on neural networks, vol. 4. Piscataway, NJ: Inst. of Electrical and Electronics Engineers; 1995. p. 1942–8.
[4] Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micro machine and human science. Piscataway, NJ: Inst. of Electrical and Electronics Engineers; 1995. p. 39–43.
[5] Fourie PC, Groenwold AA. The Particle Swarm Optimization algorithm in size and shape optimization. Struct Multidiscp Optimiz 2002;23(4):259–67.
[6] Carlisle A, Dozier G. An off-the-shelf PSO. In: Proceedings of the workshop on Particle Swarm Optimization, Indianapolis; 2001.
[7] Eberhart Russell C, Shi Yuhui. Particle Swarm Optimization: developments, applications, and resources. In: Proceedings of the 2001 congress on evolutionary computation; 2001. p. 81–6.
[8] Kennedy J, Eberhart R. Swarm intelligence. Morgan Kaufmann Publishers; 2001. ISBN: 1558605959.
[9] Schutte F, Groenwold AA. A study of global optimization using particle swarm. J Global Optimiz 2005;31:93–108.
[10] Kalivarapu V, Foo J-L, Winer E. Improving solution characteristics of Particle Swarm Optimization using digital pheromones. J Struct Multidiscip Optimiz 2007 [second review].
[11] Itzigehl PR. A method for asynchronous parallelization. In: International conference on software engineering, Proceedings of the 10th international conference on software engineering, Singapore; 1988. p. 4–9. ISBN: 0897912586.
[12] Hu X, Eberhart R, Shi Y. Engineering optimization with particle swarm. In: IEEE swarm intelligence symposium; 2003. p. 53–7.
[13] Wilke DN, Kok S, Groenwold AA. Comparison of linear and classical velocity vector update rules in Particle Swarm Optimization: notes on scale and frame invariance. Int J Numer Method Eng 2007;70:985–1008.
[14] Shi Y, Eberhart R. A modified particle swarm optimizer. In: Proceedings of the 1998 IEEE international conference on evolutionary computation. Piscataway, NJ: IEEE Press; 1998. p. 69–73.
[15] Shi Y, Eberhart R. Parameter selection in Particle Swarm Optimization. In: Proceedings of the 1998 annual conference on evolutionary computation; 1998.
[16] Natsuki H, Iba H. Particle Swarm Optimization with gaussian mutation. In: Proceedings of IEEE swarm intelligence symposium, Indianapolis; 2003. p. 72–9.
[17] Hu X, Eberhart R, Shi Y. Swarm intelligence for permutation optimization: a case study of *n*-Queens problem. In: IEEE swarm intelligence symposium, Indianapolis, IN; 2003.
[18] Venter G, Sobieszczanski-Sobieski J. Particle Swarm Optimization. AIAA J 2003;41(8):1583–9.
[19] Hu X, Eberhart R. Solving constrained nonlinear optimization problems with Particle Swarm Optimization. In: Proceedings of the sixth world multiconference on systemics, cybernetics and informatics (SCI 2002), Orlando, USA; 2002.
[20] Sedlaczek K, Eberhard P. Using augmented Lagrangian Particle Swarm Optimization for constrained problems in engineering. Struct Multidiscip Optimiz J 2006;32:277–86.
[21] Ray T, Saini P. Engineering design optimization using a swarm with an intelligent information sharing among individuals. Eng Optimiz 2001;33:735–48.
[22] Kalivarapu V, Foo J, Winer E. Implementation of digital pheromones for use in Particle Swarm Optimization. In: Proceedings of the 47th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference, Newport, RI, AIAA-2006-1917-941; 2006.
[23] Kalivarapu V, Foo J, Winer E. A parallel implementation of Particle Swarm Optimization using digital pheromones. In: Proceedings of the 11th AIAA/ISSMO multidisciplinary analysis and optimization conference, Portsmouth, VA, AIAA-2006-6908-694; 2006.
[24] Schutte J, Reinbolt J, Fregly B, Haftka R, George A. Parallel global optimization with the particle swarm algorithm. Int J Numer Method Eng 2003.
[25] Koh B, George AD, Haftka RT, Fregly B. Parallel asynchronous Particle Swarm Optimization. Int J Numer Method Eng 2006;67:578–95.
[26] Venter G, Sobieski JS. A parallel Particle Swarm Optimization algorithm accelerated by asynchronous evaluations. In: Proceedings of the sixth world congresses of structural and multidisciplinary optimization, Rio de Janeiro, Brazil; 2005.
[27] Hu X, Eberhart R, Shi Y. Particle swarm with extended memory for multiobjective optimization. In: Proceedings of 2003 IEEE swarm intelligence symposium. Indianapolis, IN, USA: IEEE Service Center; 2003. p. 193–7.
[28] Coello CC, Pulido GT, Lechuga MS. Handling multiple objectives with Particle Swarm Optimization. IEEE Trans Evol Comput 2004;8(3):256–79.
[29] Gao F, Liu H, Zhao Q, Cui G. Virus-evolutionary Particle Swarm Optimization algorithm, vol. 4222/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 156–65.
[30] Clerc M. Discrete Particle Swarm Optimization. New optimization techniques in engineering. Berlin/Heidelberg: Springer; 2004.
[31] Wang K, Huang L, Zhou C, Pang W. Particle Swarm Optimization for traveling salesman problem. In: Proceedings of the second international conference on machine learning and cybernetics; 2003.
[32] Li X, Tian P, Hua J, Zhong N. A hybrid discrete Particle Swarm Optimization for the traveling salesman problem. Lecture notes in computer science, vol. 4247/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 181–8.
[33] Shen B, Yao M, Yi W. Heuristic information based improved fuzzy discrete PSO method for solving TSP. Lecture notes in computer science, vol. 4099/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 859–63.
[34] Kitayama S, Arakawa M, Yamazaki K. Penalty function approach for the mixed discrete non-linear problems by Particle Swarm Optimization. Struct Multidiscip Optimiz 2005;32(3):191–202.
[35] Yang Q, Sun J, Zhang J, Wang C. A hybrid Particle Swarm Optimization for binary CSPs. Lecture notes in computer science, vol. 4115/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 39–49.
[36] Liu J, Sun J, Xu W. Quantum-behaved Particle Swarm Optimization for integer programming. Lecture notes in computer science, vol. 4233/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 1042–50.

[37] Tayal M, Wang B. Particle Swarm Optimization for mixed discrete, integer and continuous variables. In: Proceedings of the 10th AIAA/ISSMO multidisciplinary analysis and optimization conference, Albany, New York; 2004.
[38] Parsopoulos KE, Vrahatis MN. Recent approaches to global optimization problems through Particle Swarm Optimization. Nat Comput 2002;1:235–306.
[39] Parsopoulos KE, Vrahatis MN. On the computation of all global minimizers through Particle Swarm Optimization. IEEE Trans Evol Comput 2004;8(3):211–24.
[40] He S, Prempain E, Wu QH. An improved particle swarm optimizer for mechanical design optimization problems. Eng Optimiz 2004;36(5):585–605.
[41] Venter G, Sobieszczanski-Sobieski J. Multidisciplinary optimization of a transport aircraft wing using Particle Swarm Optimization. Struct Multidiscip Optimiz 2004;26(1–2):121–31.
[42] Pidaparti R, Jayanti S. Corrosion fatigue through Particle Swarm Optimization. AIAA J 2003;41(6).
[43] Fourie PC, Groenwold AA. The particle swarm algorithm in topology optimization. In: Proceedings of the fourth world congress of structural and multidisciplinary optimization, Dalian, China; 2001.
[44] Bochenk P, Fory's P. Structural optimization for post-buckling behavior using particle swarm. Struct Multidiscip Optimiz 2006;32(6):521–30.
[45] Schutte J, Groenwold A. Sizing design of truss structures using the particle swarms. Struct Multidiscip Optimiz 2003;25:261–9.
[46] Yang S, Huang R, Shi H. Mobile agent routing based on a two-stage optimization model and a hybrid evolutionary algorithm in wireless sensor networks. Lecture notes in computer science, vol. 4222/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 938–47.
[47] Onwubolu G, Clerc M. Optimal path for automated drilling operations by a new heuristic approach using Particle Swarm Optimization. Int J Prod Res 2004;42(3):473–91.
[48] Rameshkumar K, Suresh R, Mohanasundaram K. Discrete Particle Swarm Optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makespan. Lecture notes in computer science, vol. 3612/2005; 2005. p. 572–81.
[49] Tianzhu W, Wenhui L, Yi W, Zihou G, Dongfeng H. An adaptive stochastic collision detection between deformable objects using Particle Swarm Optimization. Lecture notes in computer science, vol. 3907/2006. Berlin/Heidelberg: Springer Publications; 2006. p. 450–9.
[50] Batkiewicz T, Dohse K, Kalivarapu V, Dohse T, Walter B, Knutzon J, et al. Multimodal UAV ground control system. In: Proceedings of the 11th AIAA/ISSMO multidisciplinary analysis and optimization conference, Portsmouth, VA, AIAA 2006-6963; 2006.
[51] Foo J, Knutzon J, Oliver J, Winer E. Three-dimensional path planning of unmanned aerial vehicles using Particle Swarm Optimization. In: Proceedings of the 11th AIAA/ISSMO multidisciplinary analysis and optimization conference, Portsmouth, VA, AIAA 2006-6995; 2006.
[52] Walter B, Sannier A, Reiners D, Oliver J. UAV swarm control: calculating digital pheromone fields with the GPU. In: The interservice/industry training, simulation & education conference (I/ITSEC), vol. 2005 (Conference Theme: One Team. One Fight. One Training Future); 2005.
[53] Gaudiano P, Shargel B, Bonabeau E, Clough B. Swarm intelligence: a new C2 paradigm with an application to control of swarms of UAVs. In: Proceedings of the eighth international command and control research and technology symposium; 2003.
[54] Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies. In: Varela F, Bourgine P, editors. Proceedings of the European conference on artificial life. Amsterdam: Elsevier; 1991.
[55] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernet Pt B 1996;26(1):29–41.
[56] Montgomery J. Towards a systematic problem classification scheme for ant colony optimization. Technical report tr02-15, School of Information Technology, Bond University, Australia; 2002.
[57] White T, Pagurek B. Towards multi-swarm problem solving in networks. In: ICMAS, third international conference on multi agent systems (ICMAS'98); 1998. p. 333.
[58] Parunak H, Purcell M, O'Conell R. Digital pheromones for autonomous coordination of swarming UAVs. In: Proceedings of first AIAA unmanned aerospace vehicles, systems, technologies, and operations conference, Norfolk, VA. AIAA; 2002.
[59] Wilke DN, Kok S, Groenwold AA. Comparison of linear and classical velocity update rules in Particle Swarm Optimization: notes on diversity. Int J Numer Method Eng 2007;70:962–84.
[60] Engelbrecht A. Fundamentals of computational swarm intelligence. Wiley Publications; 2005. ISBN: 0470091916.
[61] Web reference for test problems; 2007. <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm> [retrieved on 19.12.2007].
[62] Web reference for test problems. <http://www.geatbx.com/docu/fcnindex-01.html> [retrieved on 19.12.2007].