Journal of Systems Architecture 60 (2014) 271–279

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

A parallel Bees Algorithm implementation on GPU

Guo-Heng Luo [a], Sheng-Kai Huang [a], Yue-Shan Chang [b], Shyan-Ming Yuan [a,*]

[a] Dept. of Computer Science and Engineering, National Chiao-Tung University, 1001, University Road, Hsinchu 300, Taiwan, ROC
[b] Dept. of Computer Science and Information Engineering, National Taipei University, 151, University Road, New Taipei City 237, Taiwan, ROC

ARTICLE INFO

Article history:
Available online 10 October 2013

Keywords:
Bees Algorithm
Parallel Bees Algorithm
Swarm intelligence
GPGPU
CUDA

ABSTRACT

Bees Algorithm is a population-based method that is a computational bound algorithm whose inspired by the natural behavior of honey bees to finds a near-optimal solution for the search problem. Recently, many parallel swarm based algorithms have been developed for running on GPU (Graphic Processing Unit). Since nowadays developing a parallel Bee Algorithm running on the GPU becomes very important. In this paper, we extend the Bees Algorithm (CUBA (i.e. CUDA based Bees Algorithm)) in order to be run on the CUDA (Compute Unified Device Architecture). CUBA (CUDA based Bees Algorithm). We evaluate the performance of CUBA by conducting some experiments based on numerous famous optimization problems. Results show that CUBA significantly outperforms standard Bees Algorithm in numerous different optimization problems.

1. Introduction

Finding an optimal solution for the search problem becomes an important research question nowadays [21–23]. There are increasingly swarm intelligence [4] which is in nature the collective behavior of social animals used for finding a near optimal solution. The swarm-based optimization algorithms (SOAs) drive a search towards the optimal solution. Various algorithms, such as Ant Colony Optimization (ACO) proposed by Marco Dorigo [1], Genetic Algorithm (GA) [24], Particle swarm optimization (PSO) [3] developed by Kennedy, Artificial Bee Colony Algorithm (ABC) by proposed D. Karaboga [5], and Bees Algorithm proposed by DT Pham [2], modeled the behaviors of the swarm of animals with social organization. Self-organization is one of the system features that gets global-level response by means of many different low-level interactions.

In the SOAs, the ACO algorithm is a non-greedy population-based algorithm which emulates the behavior of real ants. The GA is based on natural selection and genetic recombination. It efficiently exploits historical information to speculate on new search areas with improved performance. The PSO is an optimization procedure based on the social behavior of groups of organizations. And the ABC is also another optimization algorithm inspired on the intelligent behavior of honey bee swarms. Bees Algorithm (BA) [2] is also a population-based method to search optimization of the problems which is inspired by the behavior of honey bees [2,6]. The algorithm performs a kind of neighborhood search combined with random search and can be used for both combinatorial optimization [25,26] and functional optimization [2]. Based on the BA, researchers have come up with several real-world applications such as data mining [7], robot controlling [8], electronic engineering [9], job scheduling [10], E-Testing [35], task allocation [36], and so on, based on Bees Algorithm.

The swarm-based optimization algorithms have been widely used to accelerate the performance of the search problems Parallelization technique is often used in the various swarm intelligence, such as a parallel implementation of ant colony optimization [27,28], parallel genetic algorithm (PGA) [29,30], parallel global optimization with the particle swarm algorithm [31], parallel Bees Algorithm (PBA) [18], and parallel artificial bee colony (PABC) algorithm [32] etc. The proposed parallelization strategy does not only degrade the quality of solutions obtained, but also achieves substantial speedup. In [28], authors discussed parallelization strategies for Ant Colony Optimization algorithms and empirically tested the simplest strategy, which of executing parallel independent runs of an algorithm. In [30], the PGA uses a mixed strategy. Subpopulations try to locate good local minima. In [31], parallel PSO performance was evaluated using two categories of optimization problems possessing multiple local minima—large-scale analytical test problems with computationally cheap function evaluations and medium-scale biomechanical system identification problems with computationally expensive function evaluations. The authors in [32] presented a parallel version of the algorithm for shared memory architectures. The entire colony of bees was divided equally among the available processors.

Graphic Processing Units (GPU) is a highly fast parallel microprocessor. There are many stream processors in a multiprocessor and each stream processor is a smallest computational unit. There is shared memory in a multiprocessor among numerous stream

* Corresponding author. Tel.: +886 910275912.
E-mail addresses: lasifu@gmail.com (G.-H. Luo), kkuume@gmail.com (S.-K. Huang), ysc.ntpu@gmail.com (Y.-S. Chang), smyuan@gmail.com (S.-M. Yuan).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.09.007

processors, they could communicate with each other by using shared memory. The GPU can accelerate computations and applications running on the CPU by loading parts of the code with high compute-loading. The NVIDIA [11,12] provides CUDA that is a general purpose parallel programming model, thus the programmers do not need to consider the complex low-level issues of GPU. Many algorithms and applications have been implemented on GPU for obtaining better performance. It supports many graphic programming APIs (Application Programming Interfaces), so developers do not have to consider more complexity of low-level problems while programming with CUDA [11,12]. Much work regarding to parallel swam intelligence algorithm has be done on GPU, such as Ant Colony Optimization [13–15], Genetic Algorithm [16,17], Particle swarm optimization [33,37], and so on. These GPU-based implementations of swarm-based optimization algorithms have proven that the GPU can be applied to significantly improve the performance of the algorithms.

Based on our knowledge, only the authors in [18] adopted Parallel Bees Algorithms (PBA) to simultaneously search the location, size and types of FACTS devices to enhance ATC between sources and sink area. Obviously, it is not generally used to solve the search problem to find the near optimal solution. In addition, in [34], authors adopted another hardware implementation (FPGA) to implement the ABC algorithm. Therefore, no attention has been paid to implement parallel Bee Algorithm on GPU yet. In order to significantly improve the performance of PBA, the objective of this paper is to design and implement a novel Parallel Bees Algorithm running on GPU. We choose CUDA framework to implement our multi-colonies Bees Algorithm on GPU called CUBA and design a new parallel multi-colonies Bees Algorithm that bring good efficiency. In the proposed algorithm, we group the threads within a block to several colonies. Each thread is assigned to a honey bee to search the solution for its colony. The proposed algorithm divides a block into different colonies by thread ID, and running Bees Algorithm independently. We evaluate the performance of CUBA by conducting some experiments based on numerous famous optimization problems. The result shows the CUBA significantly outperforms traditional BA in numerous different optimization problems.

The rest of the paper is organized as follows. Section 2 reviews the background of the Bees Algorithm and survey some related work regarding to GPU-based implementations of swarm-based optimization algorithms. Section 3 shows the methods how to parallel the Bees Algorithm to running on GPU. Section 4 evaluates and discusses the result of our experiments including the comparison of the Bees Algorithm and the CUDA-based Bees Algorithm. Finally, we give a concluding remarks and future work in the Section 5.

2. Background and related work

2.1. Bee colony optimization

In reality, there are various natural systems (i.e. social insects colonies) such as the Ants Colony and the Bees Colony in which simple individual organisms can create systems that are able to perform highly complex tasks by dynamically interacting with each other. In general, the honey bee colony consists of three kinds of adult bees: workers, drones, and a queen. Although each member in the honey bee colony has a definite task to perform, a lot of worker bees need to cooperate to complete complex jobs, such as nest building, food finding and collection, and brood rearing. In addition, individual bees (workers, drones, and queens) cannot survive without the support of the colony. Therefore, surviving and reproducing need to combine the efforts of the entire colony.

For the forage, the honey bee colony will selectively forage from nectar sources available in the field. The process is initiated by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another. Scout bees When they return to the hive, those scout bees that found a patch which is rated above a certain quality threshold deposit their nectar or pollen and go to the "dance floor" to perform a dance known as the "waggle dance". This mysterious dance is essential for colony communication, and contains three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive and its quality rating (or fitness) [2]. Authors in [2,38] summarize the concept of the Bee Colony Optimization while authors in [6,39,40] explain it in more details.

Each bee will follow a nestmate who has already discovered a patch of flowers. Upon arrival, the foraging bee takes a load of nectar and returns to the hive relinquishing the nectar to a food storer bee. After she relinquishes the food, the bee can (a) abandon the food source and become again uncommitted follower, (b) continue to forage at the food source without recruiting the nestmates, or (c) dance and thus recruit the nestmates before the return to the food source. The bee opts for one of the above alternatives with a certain probability. Within the dance area, the bee dancers "advertise" different food areas. The mechanisms by which the bee decides to follow a specific dancer are not well understood, but it is considered that "the recruitment among bees is always a function of the quality of the food source". It is also noted that not all bees start foraging simultaneously [38].

Bee Colony Optimization (BCO) [38] is a metaheuristic capable to solve difficult combinatorial optimization problems. The metaheuristic is a swarm intelligence approach, meaning it is characterized by individuals doing repetitive actions and a simple communication method between individuals, resulting in iterative improvement of solution quality. Therefore, self-organization of bees is based on a few relatively simple rules of individual insect's behavior [39]. The authors in [39] and [40] were used the collective bee intelligence in solving combinatorial optimization problems.

2.2. The Bees Colony Algorithm

The Bees Colony Algorithm [2] is a population-based method to find a near-optimal solution for the search problems. It is inspired by the behavior of honey bees in nature [2,6] and requires several parameters to be set as following: *n* (number of scout bees), *m* (number of sites selected out of *n* visited sites), *e* (number of best sites out of *m* selected sites), *nep* (number of bees recruited for best *e* sites), *nsp* (number of bees recruited for the other (m–e) selected sites), *ngh* (initial size of patches which includes site and its neighbourhood) and stopping criterion. The algorithm begins with *n* scout bees which randomly being placed in the searching domain. The basic Bees Algorithm is shown in the following subsections. The corresponded flowchart is shown in Fig. 1, detailed please refer to [2,6].

1. *Initialize populations with random solutions.*
2. *Evaluate Fitness of the population.*
3. *While (stopping criterion not met)*
4. *Select sites for neighbourhood search.*
5. *Recruit bees for selected sites (more bees for best e sites) and evaluate fitness.*
6. *Select the fittest bee from each patch.*
7. *Assign remaining bees to search randomly and evaluate their fitness.*
8. *End While*

In order to increase the search accuracy and avoid superfluous computations, Dr. Pham proposed a modified version [6], in which two new procedures were introduced as follows:

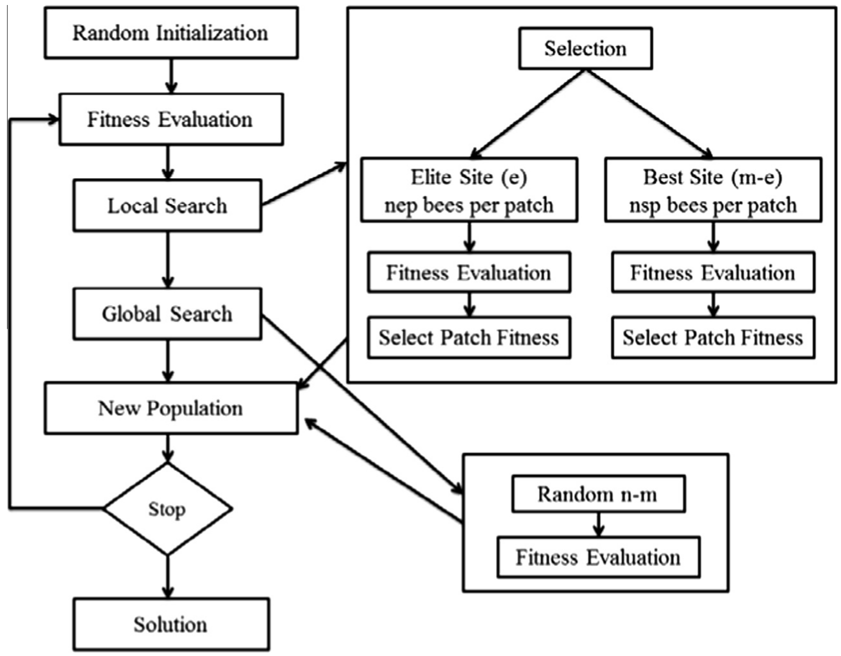


Fig. 1. Flowchart of the basic Bees Algorithm.

2.2.1. Neighbourhood shrinking

The size $a = \{a_1\ldots, a_n\}$ of the flower patches is initially set to a large value. For each variable a_i, it is set as follows.

$$a_i(t) = ngh(t) * (\max_i - \min_i)$$
$$ngh(0) = 1.0 \quad (1)$$

where t denotes the tth iteration of the Bees Algorithm main loop. The size of a patch is kept unchanged as long as the local search procedure yields higher points of fitness.

The local search is initially defined over a large neighbourhood (equal to the range of the global search), and has largely explorative feature. The local search procedure finds any better site with higher fitness, it keeps the size of *ngh* unchanged. If no improvement during the step, then the size of *ngh* be decreased. The updating formula is shown by following:

$$ngh(i+1) = 0.8 * ngh(i), \quad \textit{if no improvement}$$
$$ngh(i+1) = ngh(i), \quad \textit{else} \quad (2)$$

2.2.2. Site abandonment

When no fitness improvement after a number of times (*stlim:* limit of stagnation cycles for site abandonment) local search even by neighbourhood shrinking method, it means the local search procedure perhaps to reach the top of the local fitness peak, in other words, no further progress will be made. For efficiency, the exploration of the patch is stopped. If no better fitness of other site is generated during the remaining random search procedure then abandons this site.

Although there are several researchers come up with new models based on honeybees, our work is based on this model proposed by D.T. Pham [6].

2.3. Related work

The authors in [18] presented a parallel-based methodology for placement of Flexible AC Transmission Systems (FACTS) devices in order to reduce the time it takes to reach a solution while maximizing the available transfer capability (ATC) of a given power system. Parallel Bees Algorithms search simultaneously the location, size and types of FACTS devices to enhance ATC between sources and sink area. Results were very encouraging. In comparisons with Bees Algorithm (BA), Genetic Algorithm (GA) and Parallel Genetic Algorithms (PGA). Results show that parallel computing technique can be used effectively to reduce time to reach a solution for large scale network and FACTS devices have proven their utility for ATC improvement. However, no one implement the algorithm to be run on GPU.

On the other hand, authors in [13–17,33] have developed parallel swam intelligence algorithm on GPU. In [13], proposed a Fine-grained parallel ant colony optimization algorithm (FGACO) method based on GPU-acceleration, which maps parallel ACO algorithm to GPU through the Compute Unified Device Architecture (CUDA). The analytical results demonstrate that the proposed method increases the population size, speeds up its execution and provides ordinary users with a feasible FGACO solution. In [14], authors proposed effective parallelization strategies for the Ant Colony Optimization (ACO) metaheuristic on Graphics Processing Units (GPUs). The Max–Min Ant System (MMAS) algorithm augmented with 3-opt local search is used as a framework for the implementation of the parallel ants and multiple ant colonies general parallelization approaches. The four resulting GPU algorithms are extensively evaluated and compared on both speedup and solution quality on a state-of-the-art Fermi GPU architecture. A rigorous effort is made to keep parallel algorithms true to the original MMAS applied to the Traveling Salesman Problem. In [15], authors dealt with a GPU implementation of Ant Colony Optimization (ACO), a population-based optimization method which comprises two major stages: tour construction and pheromone update. Because of its inherently parallel nature, ACO is well-suited to GPU implementation, but it also poses significant challenges due to irregular memory access patterns. Authors contributed their work within threefold: (1) a data parallelism scheme for tour construction tailored to GPUs, (2) novel GPU programming strategies for the pheromone update stage, and (3) a new mechanism called I-Roulette to replicate the classic roulette wheel while improving GPU parallelism.

In [17], authors considered mapping of the parallel island based genetic algorithm with unidirectional ring migrations to NVIDIA

CUDA software model. The proposed algorithm begins with the input population initialization on the CPU side. Then, chromosomes and GA parameters are transferred to the GPU main memory using the system bus. Next, the CUDA kernel performing genetic algorithm on GPU is launched. In [33], authors discussed possible approaches to parallelizing PSO on graphics hardware within the CUDA, a GPU programming environment by NVIDIA which supports the company's latest cards. In particular, two different ways of exploiting GPU parallelism are explored and evaluated. In [37], a novel parallel approach to run standard particle swarm optimization (SPSO) on Graphic Processing Unit (GPU) is presented. By using the general-purpose computing ability of GPU and based on the software platform of Compute Unified Device Architecture (CUDA) from NVIDIA, SPSO can be executed in parallel on GPU.

As mentioned above, most of swarm-based optimization algorithms have been implemented to run on GPU, excepting for PBA. These implementation have proven that the GPU can be applied to significantly improve the performance of the algorithms. Based on our knowledge, only [18] adopted Parallel Bees Algorithms (PBA) to simultaneously search the location, size and types of FACTS devices. Obviously, it is not generally used to solve the search problem to find the near optimal solution. In addition, in [34], authors adopted another hardware implementation (FPGA) to implement the ABC algorithm. It is obviously there are not any research to design and implement Parallel Bees Algorithm on GPU yet.

3. Parallel Bees Algorithm on GPU

The major key of deciding the accelerated effect is the level of parallelization. In standard Bees Algorithm, most computational loads are in the neighbourhood search procedure. A naïve method is to take the neighbourhood search procedure as a kernel to distribute the computations in loop of the procedure. In fact, the optimal number of the neighbourhood size is fluctuant according to different features of functions. However, if the size of the neighbourhood is not larger than the number of total threads within the GPU, then the accelerated effect would not be obvious. Another common solution is "multi-colonies" that means we should run many Bees Algorithms independently in each threads. There are two major disadvantages. The first is that each thread contains many conditional branch. Obviously, it is difficult to avoid divergent branch, so the overhead would be too expensive. The second one is that the communication among the threads after a round would be more complex to do. For the reasons above, we design a new Bees Algorithm of parallel multi-colonies that bring good efficiency.

3.1. System overview

CUDA framework is used to implement our multi-colonies Bees Algorithm on GPU called CUBA. Fig. 2 shows an overview of CUBA framework. In CUBA framework threads within one block are classified into several colonies. In other words, each thread is assigned to a honey bee to search the solution for its colony. A block is divided into different colonies indexed by thread ID, and running Bees Algorithm independently. When one iteration finished, the CUBA change the information between colonies in the same block by using shared memory. Since the communication latency between colonies is critical for converge time of the algorithm, the information sharing between threads does not adopt global memory. In other words, the colonies will not communicate with each other if they are in different blocks because the shared memory is shared by threads in the same block.

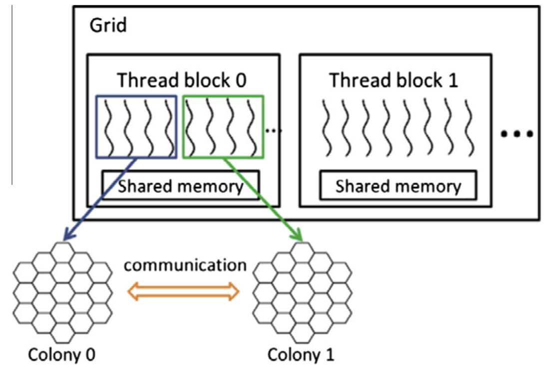


Fig. 2. Framework of the CUBA.

3.2. Parallelization algorithm

This section describes the parallel Bees Algorithm on CUDA in detail. To modify standard BA into parallel BA and run it on CUDA, we need to overcome many implementation issues. This section we will depict the design idea of CUDA based BA, as shown in Fig. 3, and examine the implementation issues in detail. Following we will explain the steps of the algorithm and corresponding implementation issues.

3.2.1. Parallel initialization

In BA approach, the initialization of population and the evaluation of the fitness of the population will be achieved one-by-one. In the parallel BA, naturally, it can be done in parallel. Hence, CUBA will distribute and compute them through parallel threads in GPU. In the first step, the CUBA need to setup required parameters, allocate and initiate shared memory for communication, configure the number of colony, and so on. These are similar with the standard BA but for parallel BA. In addition, the algorithm also needs to randomly generate new sites and compute each bee's fitness in parallel.

3.2.2. Odd–even sort

It is necessary to sort the fitness of all populations to get the best *m* sites. Because the size of the sorting data in this application is small respect to others, we sort the colonies in the same block individually by using Odd–Even Sorting algorithm [19,20] that is based on the Bubble Sort technique of comparing two elements and switching them by the comparing rule. Although there are a variety of well-known sorting algorithms can be utilized, this method only requires n/2 iterations of the two phase sort.

3.2.3. Group Bees into different colonies

We divide threads in the blocks to different colonies according to their thread ID, each thread is assigned to a honey bee and searching the solution for its colony, so there are a number of colonies run Bees Algorithm in parallel. The number of bees and colonies in the algorithm is depending on what the number of blocks per grid and number of threads per block we set.

The number of colonies in a block = number of threads per block / number of bees per colony.

3.2.4. Modified Bees Algorithm

In order to parallelize the standard BA to be parallel, some implementation issues need to take into account, as follows.

3.2.4.1. Modification of local search. The local search in traditional BA approach, more bees (*nep*) recruit for elite sites and fewer bees (*nsp*) recruits for the rest of sites from *e* sites. It is reasonable

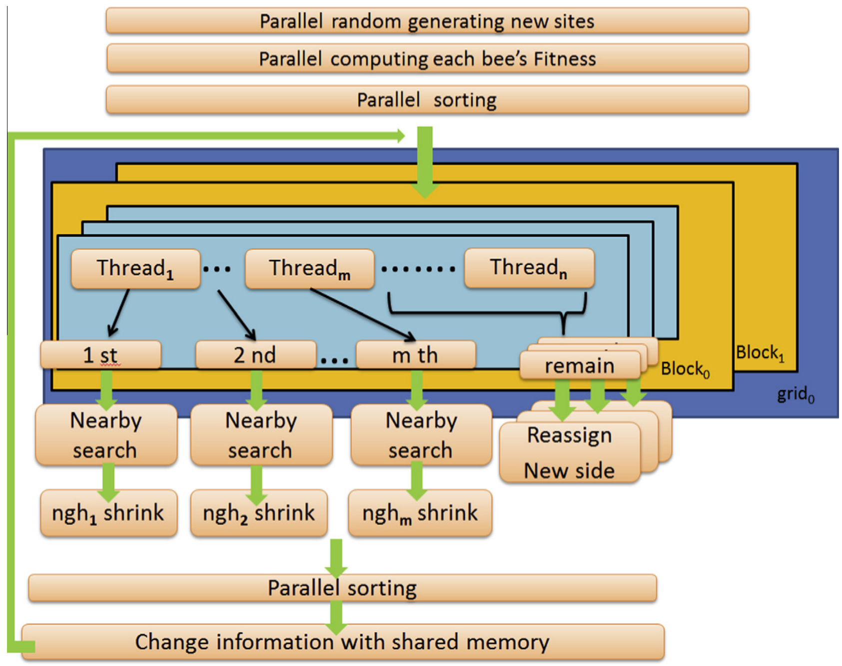


Fig. 3. Algorithm of the CUBA.

because the mechanism is based on probability. But in our system, we just assign *nep* bees to recruit *m* sites for balancing the loading among the threads, to be more precisely, it does not make sense in parallel architecture if some threads would do nothing after finishing their jobs and waiting for the others.

3.2.4.2. Random seeds. We have different threads in GPU with different random seeds. It can benefit the performance by generating random numbers from different sequences in parallel.

3.2.4.3. Neighbourhood shrinking. According to the new procedure "neighbourhood shrinking" in BA, *ngh* constantly change values, in our approach, we have numerous local searching in different sites simultaneously for parallelism, and we let the recruited bees in different sites with different *ngh*. Another adjust is that we do not need to set a such large number of the recruited bees like in BA, because we have so numerous colonies to search simultaneously that the risk which may cause wrong shrink we accept is much lower. In the meanwhile, the rapid decreasing of *ngh* could bring a faster converge time. What shrinking equation we use is the same with the equation in the Bees Algorithm. Initially, the size of *ngh* is set to a large value.

3.2.4.4. Communication with shared memory. In general parallel architectures may use shared memory or message passing method to communicate between the multiple processing units. There is a shared memory in the same block in CUDA architecture, so we use it to implement the communication in the end of the each iteration. In this strategy, there are three issues we have to concern. The first is what information to share, the second one is who to share with, and the last is how long to communicate once.

We had tried and compared several mechanisms for communication. For example, we sort the best results which are gained from individual colonies in the same block after neighbourhood search, and sharing the best to others. To explain in detail, the site with lowest fitness in each colony is replaced by best one with highest fitness in the block. The result shows that converge rate is quite good. However, a sorting procedure often makes an impact on the execution time, finally, we develop the two-phase communication that avoiding sorting and with good converge rate, too. The paired exchange take few time to share, and the second phase improve the global convergence over time. The method is shown in Fig. 4.

The two kinds of communication are executed alternately one after one iteration.

4. Experimental results and analysis

To evaluate the performance of the proposed algorithm, we conducted some experiments to evaluate and compare both of the execution time with CUDA on GPU and the execution time with C++ on CPU to verify efficiency of the algorithm. The configuration of the evaluation platform are shown as Table 1 and described as flows: We adopt AMD Athlon (tm) II and GeForce GTX 460 for our computation platform. The host is AMD Athlon(tm) II which has 4 cores, and each core has clock rate with 3.0 GHz. The device is GeForce GTX 460 which has 7 multiprocessors (MPs) and each MP has 48 CUDA cores. Totally, there are 336 CUDA cores in the device.

4.1. Benchmark functions

Table 2 shows the equations of 9 continuous function minimization benchmarks [6]. The equations are given together with the range of the variables and global minimum. These functions are widely used multi-modal test functions.

4.2. Analysis and result

For CUBA, there are 5 parameters we have to set, *GridDim*, *BlockDim*, *N*, *M* and *nep*. In the CUDA programming, the code running parallel is called "kernel", the job size of kernel is so called "grid". The programmer should set a dimensional number of grid. CUBA will divide the job to many smaller jobs and distribute them to different multiprocessors to execute. The size of each smaller job is called "block". As setting dimensional number of grid, the

Adjacent exchange (first phase):

If *colony ID is odd*, then

exchange with colony (ID+1) % number of colonies per block

If *colony ID is even*, then

exchange with colony (ID-1) % number of colonies per block

Skip one exchange phase (second phase)

If *colony ID is odd*, then

exchange with colony (ID+2) % number of colonies per block

If *colony ID is even*, then

exchange with colony (ID-2) % number of colonies per block

Fig. 4. Two-phase communication mechanism.

Table 1
Hardware configurations.

Device	CPU	GPU
Processor	AMD Athlon(tm) II x4 disable 3 cores	GeForce GTX 460
Number of cores	4 cores	336 cores (7 MPs)
Clocks	3.0 GHz	675 MHz
Memory	DDR3–1333	GDDR5
Memory size	4 GB	512 MB
OS	Win7(32 bit)	
Compute capability	–	2.1
CUDA version	–	4.1

Table 2
Benchmark functions.

Function	Equation	Minimum
Ackley (2D)	$f(x1,x2)=20-20e^{-0.2\sqrt{\frac{1}{2}(x1^2+x2^2)}}-e^{\frac{1}{2}[\cos(2\pi x1)+\cos(2\pi x2)]}+e,-32<x1<x2$	$\vec{x}=(\vec{0})f(\vec{x})=0$
Easom (2D)	$f(x_1,x_2)=-\cos(x_1)*\cos(x_2)e^{-(x_1-\pi)^2-(x_2-\pi)^2},-100<x_1<100$	$\vec{x}=(\pi,\pi)f(\vec{x})=-1$
Goldstein and Price (2D)	$A(x_1,x_2)=1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)B(x_1,x_2)$ $=30+(2x_1+3x_2)^2(18-32x_1+12x_1^2-48x_2+36x_1x_2+27x_2^2)f(x_1,x_2)=AB,-2<x_1<s$	$\vec{x}=(0,-1)f(\vec{x})=3$
Martin and Gaddy (2D)	$f(x_1,x_2)=(x_1-x_2)^2+\left[\frac{(x_1+x_2-10)}{3}\right]^2,-20<x_i<20$	$\vec{x}=(5,5)f(\vec{x})=0$
Schaffer (2D)	$f(x_1,x_2)=0.5+\frac{\left[\sin(\sqrt{x_1^2+x_2^2})\right]^2-0.5}{\left[1.0+0.001(x_1^2+x_1^2)\right]^2},-100<x_i<100$	$\vec{x}=(\vec{0})f(\vec{x})=0$
Schwefel (2D)	$f(x_1,x_2)=-x_1\sin\lvert(\sqrt{\lvert x_1\rvert})--x_2\sin(\sqrt{\lvert x_2\rvert}),-500<x_i<500$	$\vec{x}=(420\vec{.}97)f(\vec{x})=-837.97$
Hyper Sphere (10D)	$f(\vec{x})=\sum_{i=1}^{10}x_i^2,-100<x_i<100$	$\vec{x}=(\vec{0})f(\vec{x})=0$
Griewank (10D)	$f(\vec{x})=\frac{1}{4000}\sum_{i=1}^{10}(x_i-100)^2-\prod_{i=1}^{10}\cos\left(\frac{x_i-100}{\sqrt{i+1}}\right)+1,-600<x_i<600$	$\vec{x}=(1\vec{0}0)f(\vec{x})=0$
Rosenbrock (10D)	$f(\vec{x})=\Sigma_{i=1}^{9}100(x_{i+1}-x_i^2)^2+(1-x_i)^2,-50<x_i<50$	$\vec{x}=(\vec{0})f(\vec{x})=0$

programmer has to set the dimensional number of block, meaning how many threads in a block. In our algorithm, there are *BlockDim*/N colonies per block. For example if we set *BlockDim* = 256 and *N* = 8, then there are 32 colonies in a block. The parameters are set according to the results of the experiments, we will discuss in more detail later.

All the programs both of BA and CUBA in the following experiments were run until either the minimum of the function was approximated to better than 0.001, or reached a maximum number of cycles (here we set 5000). In the BA version, because there is only one colony foraging, if it make a wrong *ngh* shrinking, the global optimum solution will never be found. To overcome this, BA set a quite large *nep* and *nsp* to avoid as possible. Ideally, the CUBA version has more colonies foraged parallel in the same time, so we can afford more risks that we making a wrong *ngh* shrinking procedure. To prove this assumption, we test the 9 functions with various *nep*, 1, 2, 4, 8, 16 and 32. At first we set *GridDim* = 4, *BlockDim* = 256. It is a reasonable number of *BlockDim*. In most of GPUs, there are 32 or 64 stream processors (the smallest computation unit) in a multiprocessor, so we take the number as multiple of number of stream processors.

When we found adaptive *nep* for each function, we tried to decrease the number of *BlockDim*, that means the number of colonies is decreased. Another issue is what will happen if we increase the *N* and *BlockDim* with the same factor, in other words, we increase the bees for every colony and fix the number of colonies in a block. Finally, we will use the best parameters set we found from the three experiments, and take the result to compare with the Bees Algorithm.

4.2.1. Analysis of nep

First we vary the *nep* to evaluate the execution time and the amount of iteration before the benchmark functions convergence. The result shows in Table 3. For low dimensional functions, we only need very small *nep* (about 1 or 2) to get a good solution with less time. But for high dimensional functions, we need bigger *nep*, too small *nep* will lead the solution converge to the number with

Table 3
nep increasing (execution time and amount of iteration).

Benchmark functions	*nep* = 1		*nep* = 2		*nep* = 4		*nep* = 8		*nep* = 16		*nep* = 32	
	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations
Ackley(2D)	8.13	38	7.51	25	9.50	24	14.28	26	19.87	22	42.35	26
Easom(2D)	6.17	27	6.81	25	8.03	23	14.28	26	15.04	20	25.07	20
Goldstein and Price(2D)	6.01	22	5.56	12	7.57	17	9.14	14	12.92	13	18.49	11
Martin and Gaddy(2D)	4.58	17	4.72	16	4.92	14	5.67	15	5.52	8	5.74	5
Schaffer(2D)	7.11	29	7.27	22	9.24	22	11.81	19	16.20	16	33.28	20
Schwefel(2D)	5.52	27	6.35	30	6.48	22	8.33	23	11.18	21	16.20	19
HyperSphere (10D)	x	x	x	x	12	41	17.12	42	32.95	52	63.10	59
Griewank (10D)	x	x	52.51	51	70.95	43	125.23	44	219.76	42	439.90	44
Rosenbrock (10D)	x	x	x	x	x	x	1796.32	439	3140.61	410	4857.48	329

Table 4
Colonies increasing.

Benchmark functions	*blockDim* = 32		*blockDim* = 64		*blockDim* = 128		*blockDim* = 256		*blockDim* = 512		*blockDim* = 768	
	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations	Time (ms)	iterations
Ackley(2D)	6.89	40	6.05	31	6.76	32	8.13	38	8.341	27	12.66	30
Easom(2D)	x	X	5.27	31	6.02	32	6.28	29	7.76	26	10.88	25
Goldstein and Price (2D)	5.73	32	5.01	24	7.25	19	6.01	22	7.08	28	9.99	17
Martin and Gaddy (2D)	3.83	24	3.27	7	4.17	14	4.58	17	5.43	12	8.33	17
Schaffer(2D)	x	x	6.32	33	7.02	33	7.11	29	7.85	22	10.32	17
Schwefel (2D)	4.90	34	5.03	35	5.29	32	5.52	27	7.08	28	9.89	24
HyperSphere(10D)	19.71	67	22.36	56	15.79	42	17.12	42	20.15	41	25.45	41
Griewank (10D)	55.58	61	50.01	54	48.02	49	52.51	51	61.75	52	80.98	51
Rosenbrock (10D)	x	x	x	x	810.83	177	1796.32	439	432.984	94	598.962	102

Table 5
Bees increasing.

Benchmark functions	*blockDim* = 128 *N* = 4		*blockDim* = 256 *N* = 8		*blockDim* = 512 *N* = 16	
	Time (ms)	Iterations	Time (ms)	Iterations	Time (ms)	Iterations
Ackley	6.76	36	8.13	38	9.68	33
Easom	5.72	31	6.28	29	7.87	26
Goldstein and price	5.35	23	6.01	22	7.13	17
Martin and Gaddy	4.11	22	6.01	22	6.00	19
Schaffer	6.76	18	6.35	23	8.72	26
Schwefel	4.70	26	5.52	27	7.61	30
HyperSphere	15.88	46	17.12	42	25.12	49
Griewank	53.62	57	52.51	51	65.10	53
Rosenbrock	x	x	1796.32	439	2166.75	463

big error. In addition, the solution with less execution time does not mean it is with less iterations.

4.2.2. Analysis of the number of colonies

Second, we vary the number of colonies to evaluate the execution time and the amount of iteration before the benchmark functions convergence. The result is shown in Table 4, most of the functions get good performance and fewer execution time with small number of *BlockDim*, excluding the three high dimensional functions, such as *HyperSphere(10D)*, *Griewank(10D), and Rosenbrock(10D)*. For these high dimensional function, small number of *BlockDim* not always bring the benefit, the best number of *BlockDim* for *HyperSphere* and *Griewank* are 128, and 512 for *Rosenbrock*. Similarly, the solution with less execution time does not mean it is with less iterations.

4.2.3. Analysis of the number of bees

Second, we vary the number of bees to evaluate the execution time and the amount of iteration before the benchmark functions

Table 6
Combinations of The Bees Algorithm parameters.

Benchmark	*n*	*m*	*e*	*nep*	*nsp*	*stlim*
Ackley	30	8	1	20	10	5
Easom	20	14	1	30	5	10
GoldsteinAndPrice	10	4	2	30	10	10
MartinAndGaddy	10	7	1	30	10	10
Schaffer	10	4	2	30	10	10
Schwefel	20	14	1	30	5	10
HyperSphere	10	4	2	30	10	10
Griewank	20	18	1	10	5	5
Rosenbrock	10	4	2	30	10	10

Table 7
Combinations of CUDA Bees Algorithm parameters.

Benchmark	*GridDim*	*BlockDim*	*n*	*m*	*nep*
Ackley	4	64	8	6	1
Easom	4	64	8	6	1
GoldsteinAndPrice	4	64	8	6	1
MartinAndGaddy	4	64	8	6	1
Schaffer	4	64	8	6	1
Schwefel	4	32	8	6	1
HyperSphere	4	128	8	6	8
Griewank	4	128	8	6	2
Rosenbrock	4	512	8	6	8

convergence. Table 5 also show that we could set smaller number of *BlockDim* and *N* in low dimensional functions. In high dimensional function, for some functions, we could set a small bees number for shorter execution time like *HyperSphere* and *Griewank*. But sometimes the number of bees could not be too small, or the solution will converge with big error like *Rosenbrock*.

4.2.4. Robustness and Speedup

We calculated the execution times and the success rates of fifty running times for the two algorithms. For estimating the execution time of BA, the parameters for all benchmark functions are given in Table 6 according to the original set in the paper [6], and the Table 7 shows the parameters set of CUBA by using the best parameters set we have found before.

We also evaluated the success rate of those functions with the error and got 100% success rates by using both of the algorithms in 50 times in the standard Bees Algorithm and CUDA-based BA respectively. It shows that the proposed CUBA can obtain the same success rate compared with standard BA. Finally, we evaluate the speedup in terms of execution time and executed iteration between them. Since CUDA supports fast math library, we are encouraged to use them as often as possible. The evaluation results are shown in Table 8 for execution time speedup and for executed iteration speedup. It is obviously not only the execution time of CUBA less than BA, but also less iterations to be executed. The result shows that the proposed CUDA-based BA has 13 $\sim$56 times faster than BA in various benchmark functions. As the result in Table 9, CUBA takes less iterations to find the solutions. It makes huger difference while running the high dimensional functions.

Table 8
Speedup (time).

Benchmark Functions	Standard BA	CUDA-based BA(CUBA)	Speedup
Ackley	273 ms	6.05 ms	45.12
Easom	70 ms	5.27 ms	13.28
Goldstein and Price	92 ms	5.01 ms	18.36
Martin and Gaddy	76 ms	3.27 ms	27.24
Schaffer	231 ms	6.32 ms	36.55
Schwefel	279 ms	4.90 ms	56.93
HyperSphere	389 ms	15.79 ms	24.63
Griewank	2520 ms	48.02 ms	52.47
Rosenbrock	5595 ms	432.98 ms	12.92

Table 9
Iteration reduction ratio (# of Standard BA/ # of CUBA).

Benchmark Functions	Standard BA	CUDA-based BA (CUBA)	Iteration reduction factor
Ackley	90	31	2.90
Easom	44	31	1.41
Goldstein and Price	49	24	2.04
Martin and Gaddy	50	7	7.14
Schaffer	55	33	1.66
Schwefel	141	34	4.14
HyperSphere	158	42	3.76
Griewank	1487	49	30.34
Rosenbrock	4456	94	47.40

5. Conclusion and future work

In this paper, first we have proposed Parallel Bees Algorithm based on CUDA. We modify the local search procedure. Running in SIMT (Single Instruction Multiple Threads) hardware architecture, we merge the two parts of the local searching sites avoiding wasting the computing powers of GPU. For the same reason, we have no site abandonment procedure. In addition, we let the bees recruiting in different sites maintain own *ngh*, meaning they shrink independently. We sort the colonies in the same block individually by using Odd–Even Sorting algorithm to get the benefit of parallel. The communication mechanism between colonies in the same block is also important point to decrease the convergence time, in our algorithm, we choose two-phase communication for better result.

To find the features of this new algorithm, we modify the parameters, and get the result of the most of low dimensional functions could be run with good performance by using small *nep*. That is one of key points why CUBA runs with faster convergence time than standard Bees Algorithm. We also try to decrease the number of colonies in a CUDA block and decrease the number of bees per colony to optimize the parameters set for each functions. We also compared the convergence time (error < 0.001) of CUDA Bees Algorithm with The Bees Algorithm. The experimental result shows CUBA is faster than BA at least 13 times in 9 different functions of optimization problems.

In the future, we will compare the CUBA to other parallel swarm based algorithm, and try more parallel sorting algorithm and communication mechanism. Not only for solving the optimization problem, we would also apply the proposed algorithm to real world applications. Today, cloud computing becomes more and more important and popular. There are some platforms which provide GPU based cloud computing service. We would improve the proposed algorithm and testing in GPUs clusters environment.

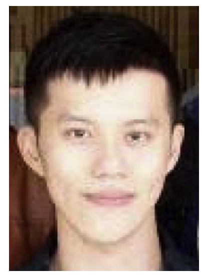
Acknowledgements

We are grateful for the many excellent comments and suggestions made by the anonymous referees. This work was supported in part by the Nation Science Council of Republic of China under Grant no. NSC 101-2221-E-009-034-MY2.

References

[1] M. Dorigo, Optimization, Learning and Natural Algorithms, Ph.D. thesis, Politecnico di Milano, Italie, 1992.
[2] D.T. Pham, E. Koc, A. Ghanbarzadeh, S. Otri, S. Rahim, M. Zaidi, The Bees Algorithm–a novel tool for complex optimisation problems, in: Proceedings of the Second International Virtual Conference on Intelligent Production Machines and Systems, 2006, pp. 454–461.
[3] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. IV, 1995, pp. 1942–1948.
[4] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, New York, 1999.
[5] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Global Optimization 39 (3) (2007). pp. 459–171.
[6] D.T. Pham, M. Castellani, The Bees Algorithm: modelling foraging behaviour to solve continuous optimization problems, Proceeding of Institute Mechanical Engineering, C: Journal of Mechanical Engineering and Science 223 (12) (2009) 2919–2938.
[7] D.T. Pham, S. Otri, A. Afify, M. Mahmuddin, H. Al-Jabbouli, Data clustering using the Bees Algorithm, in: Proceedings of the 40th CIRP International Manufacturing Systems, Seminar, 2007.

[8] D.T. Pham, A.H. Darwish, E.E. Eldukhri, S. Otri, Using the Bees Algorithm to tune a fuzzy logic controller for a robot gymnast, in: Proceedings of International Conference on Manufacturing Automation, 2007, pp. 28–30.
[9] K. Guney, M. Onay, Bees Algorithm for design of dual-beam linear antenna arrays with digital attenuators and digital phase shifters, International Journal of RF and Microwave Computer-Aided Engineering 18 (4) (2008) 337–347.
[10] D.T. Pham, E. Koc, J.Y. Lee, J. Phrueksanant, Using the Bees Algorithm to schedule jobs for a machine, in: Proc Eighth International Conference on Laser Metrology, CMM and Machine Tool Performance, LAMDAMAP, Euspen, 2007, pp. 430–439.
[11] NVIDIA CUDA Programming Guide Version 4.2: NVIDIA Corporation, 2012.
[12] NVIDIA CUDA Best Practices Guild, 4.2 edition, NVIDIA Corporation, 2012.
[13] Jianming Li, Xiangpei Hu, Zhanlong Pang, Kunming Qian, A parallel Ant colony optimization algorithm based on fine-grained model with CPU-acceleration, International Journal of Innovative Computing, Information and Control, 5 11(A) (2009) 3707–3716.
[14] Audrey Delévacq, Pierre Delisle, Marc Gravel, Michaël Krajecki, Parallel ant colony optimization on graphics processing units, Journal of Parallel and Distributed Computing 73 (1) (January 2013) 52–61.
[15] José M. Cecilia, José M. García, Andy Nisbet, Martyn Amos, Manuel Ujaldón, Enhancing data parallelism for Ant Colony Optimization on GPUs, Journal of Parallel and Distributed Computing, 73(1) (2013) 42–51.
[16] W.B. Langdon, Graphics processing units and genetic programming: an overview, Soft Computing 15 (8) (August 2011) 1657–1669.
[17] Petr. Pospichal, Jiri. Jaros, Josef. Schwarz, Parallel genetic algorithm on the CUDA architecture, Lecture Notes in Computer Science 6024 (2010) 442–451.
[18] A.K.R. Mohamad Idris, M.W. Mustafa, A Parallel Bees Algorithm for ATC enhancement in modern electrical network, in: 2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer, Simulation, 2010, pp. 450–455.
[19] S. Lakshmivarahan, S.K. Dhall, L.L. Miller, L. Alt Franz, C. Marshall, Yovits, (Eds.), Parallel Sorting Algorithms, Advances in Computers, vol. 23, Academic Press, 1984, pp. 295–351.
[20] M. Phillips. Available from: <http://homepages.ihug.co.nz/~aurora76/Malc/Sorting_Array.htm#Exchanging>.
[21] M. Kai, T. Hatori, Parallelized search for the optimal/sub-optimal solutions of task scheduling problem taking account of communication overhead, in: 2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, vol. 1, pp. 327–330.
[22] C. Nakazawa, S. Kitagawa, Y. Fukuyama, Hsiao-Dong Chiang, A method for searching multiple local optimal solutions of nonlinear optimization problems, in: 2005. IEEE International Symposium on Circuits and Systems, ISCAS, vol. 5, 2005, pp. 4907–4910.
[23] I.R. De Pablo, A. Becker, T. Bretl, An optimal solution to the linear search problem for a robot with dynamics, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 652–657.
[24] D.E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Reading: Addison-Wesley Longman, 1989.
[25] Duc Truong Pham, Ashraf Afify, Ebubekir Koc, Manufacturing cell formation using the Bees Algorithm, in: IPROMS 2007 Innovative Production Machines and Systems Virtual Conference, Cardiff, UK.
[26] D.T. Pham, E. Koç, J.Y. Lee, J. Phrueksanant, Using the Bees Algorithm to schedule jobs for a machine, in: Proc Eighth International Conference on Laser Metrology, CMM and Machine Tool Performance, LAMDAMAP, Euspen, UK, Cardiff, 2007, p. 430–439.
[27] Marcus. Randalla, Andrew. Lewisb, A parallel implementation of ant colony optimization, Journal of Parallel and Distributed Computing 62 (9) (2002) 1421–1432.
[28] Thomas. Stützle, Parallelization strategies for ant colony optimization, Lecture Notes in Computer Science 1498 (1998) 722–731.
[29] Chrisila B. Pettey, Michael R. Leuze, John J. Grefenstette, A parallel genetic algorithm, in: The Second International Conference on Genetic Algorithms on Genetic algorithms and their application, 1987, pp. 155–161.
[30] H. Mühlenbein, M. Schomisch, J. Born, The parallel genetic algorithm as function optimizer, Parallel Computing 17 (6–7) (1991) 619–632.
[31] J.F. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka, A.D. George, Parallel global optimization with the particle swarm algorithm, International Journal for Numerical Methods in Engineering 61 (13) (2004) 2296–2315.
[32] H. Narasimhan, Parallel artificial bee colony (PABC) algorithm, in: 2009 World Congress on Nature & Biologically Inspired Computing, 2009, pp. 306–311.
[33] Luca Mussi, Fabio Daolio, Stefano Cagnoni, Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture, Information Sciences 181 (20) (2011) 4642–4657.
[34] D.M. Munoz, C.H. Llanos, L.D.S. Coelho, M. Ayala-Rincon, Accelerating the artificial bee colony algorithm by hardware parallel implementations, in: 2012 IEEE Third Latin American Symposium on Circuits and Systems (LASCAS), March 2 2012, pp. 1–4.
[35] P. Songmuang, M. Ueno, Bees Algorithm for construction of multiple test forms in E-Testing, IEEE Transactions on Learning Technologies 4 (3) (2011) 209–221.
[36] A. Jevtic, A. Gutierrez, D. Andina, M. Jamshidi, Distributed Bees Algorithm for task allocation in swarm of robots, IEEE Systems Journal, 6(2) (2012) 296–304.
[37] You Zhou, Ying Tan, GPU-based parallel particle swarm optimization, IEEE Congress on Evolutionary Computation, 2009, CEC '09, pp. 1493–1500.
[38] Dusan Teodorovic, Panta Lucic, Goran Markovic, Mauro Dell' Orco, Bee colony optimization: principles and applications, in: 8th Seminar on Neural Network Applications in Electrical Engineering, NEUREL-2006, Serbia, September 25–27, 2006, pp. 151–156.
[39] Scott Camazine†, James Sneyd, A model of collective nectar source selection by honey bees: self-organization through simple rules, Journal of Theoretical Biology, 149(4) 21 (April 1991) 547–571.
[40] P. Lu, D.Teodorovi, Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence, in: Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis, Sao Miguel, Azores Islands, Portugal, 2001, pp. 441–445.



Sheng-Kai Huang received his Master Degree from Institute of Computer Science and Engineering of National Chiao Tung University in 2012. He is now with Telecommunication Laboratories Chunghwa Telecom Co., Ltd. His research interests are in Parallel, Cloud and Mobile Computing.

Guo-Heng Luo received his Master Degree from Institute of Computer Science and Engineering of National Chiao Tung University in 2009. He is now a Ph.D. student with the Institute of Computer Science and Engineering, National Chiao Tung University. His research interests are in Web 2.0, Parallel and Cloud Computing.

Yue-Shan Chang received his PhD Degree from Computer and Information Science at the National Chiao Tung University in 2001. He joined the Department of Electronic Engineering of the Ming Hsing University of Science and Technology in August 1992. Since August 2004, he joined the Department of Computer Science and Information Engineering, National Taipei University, Taipei County, Taiwan. Since August 2010, he had been a Professor. His research interests are in distributed systems, web service composition, information retrieval, mobile computing and grid computing.

Shyan-Ming Yuan received his BSEE degree from National Taiwan University in 1981, his MS degree in Computer Science from University of Maryland, Baltimore County in 1985, and his PhD degree in Computer Science from the University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he has been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became the Professor in June 1995. His current research interests include Distributed Objects, Internet Technologies, and Software System Integration. Dr. Yuan is a member of ACM and IEEE.