International Journal of Engineering

Journal Homepage: www.ije.ir

Parallel Implementation of Particle Swarm Optimization Variants Using Graphics Processing Unit Platform

S. Jam, A. Shahbahrami*, S. H. S. Ziyabari

Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran

PAPER INFO

Paper history:
Received 23 June 2016
Received in revised form 20 September 2016
Accepted 11 November 2016

Keywords:
Particle Swarm Optimization
Adaptive Particle Swarm Optimization
Particle Swarm Optimization with an Aging Leader and Challengers
Graphics Processing Unit

ABSTRACT

There are different variants of Particle Swarm Optimization (PSO) algorithm such as Adaptive Particle Swarm Optimization (APSO) and Particle Swarm Optimization with an Aging Leader and Challengers (ALC-PSO). These algorithms improve the performance of PSO in terms of finding the best solution and accelerating the convergence speed. However, these algorithms are computationally intensive. The goal of this paper is high performance implementations of Traditional PSO (TPSO), APSO and ALC-PSO using CUDA technology. We have implemented these three algorithms on both central processing unit (CPU) and graphics processing unit (GPU) in order to analyze and improve their performance and reduce their computational times. We have achieved speedups up to 14.5x, 31x, and 152x, for GPU-TPSO , GPU-ALCPSO , and GPU-APSO, respectively. In addition, different number of threads has been chosen in order to find an appropriate number of threads per block for both APSO and ALC-PSO algorithms. Our experimental results show that the best choice for number of threads per block depends on the number of existing variables and constants in each algorithm and the number of registers per multiprocessor.

doi: *10.5829/idosi.ije.2017.30.01a.07*

1. INTRODUCTION

Particle Swarm Optimization (PSO) has been successfully applied to a lot of difficult and complex optimization problems such as artificial neural network training, function optimization and fuzzy system control [1-3]. However, PSO is a population-based iterative algorithm and similar to most of the evolutionary algorithms, it is computationally intensive. The main reason is that optimizing process of PSO requires a large number of fitness evaluations which runs sequentially on CPU. So, a result, running speed of PSO may become quite slow [4]. In addition, PSO cannot overcome the problem of premature convergence that is a big disadvantage [5].

Researchers who have focused on PSO have considered accelerating convergence speed and avoiding the local optima as their major goals [6]. A number of variant PSO algorithms have been proposed [5, 7, 8], but most of them have only focused on one of these goals in order to improve the performance of PSO, whereas Adaptive PSO (APSO) [6] and the PSO with an Aging Leader and Challengers (ALC-PSO) [9] have been proposed to achieve both of these goals. In this article, the original PSO is called Traditional Particle Swarm Optimization (TPSO). However, the improved PSO algorithms are also computationally intensive [3]. We have also shown that TPSO, APSO and ALC-PSO take so much time. For example the execution times of the APSO algorithm using 2000 particles for three benchmark test functions are 3535, 3773 and 4589 seconds, respectively.

GPU pipelines are used as a coprocessor for implementing highly parallelizable algorithms, since they became programmable and the programming model was unified by the NVIDIA CUDA architecture [1][10]. The main goal of this paper is high performance implementation of PSO, APSO and ALC-

*Corresponding Author's Email: shahbahrami@guilan.ac.ir

1.CUDA, C programming guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide. (2014).

Please cite this article as: S. Jam, A. Shahbahrami, S. H. S. Ziyabari, Parallel Implementation of Particle Swarm Optimization Variants Using Using Graphics Processing Unit Platform, International Journal of Engineering (IJE), TRANSACTIONS A: Basics Vol. 30, No. 1, (January 2017) 48-56

SPO on GPU platform. In order to obtain high performance in comparison with to CPU implementation, we have exploited different parallelisms such as loop-level parallelism, and have achieved significant speedup of up to 152x faster than CPU based implementation. Our main contributions compared to other related works are as follows:

- Analyzing and implementing TPSO, APSO and ALC-PSO on CPU platform in order to show that they are computationally intensive. Experimental results show that APSO is the slowest algorithm and it takes more than one hour for 2000 particles using some benchmark test functions.
- Exploiting different available parallelism such as loop-level parallelism in these three algorithms using GPU characteristics in order to improve their performance.
- Our experimental results show that the maximum speedup is obtained for APSO algorithm which is up to 152x.
- Using multiple threads in order to calculate the ideal occupancy –number of threads per block- for parallel implementations of APSO and ALCPSO algorithms.
- Evaluating the relation among number of threads, number of existing variables and constants in algorithms and number of registers per multiprocessor which can help us to choose the best number of threads per block in order to achieve highest speedup.

This paper is organized as follows. Background information and related works are discussed in Section 2. In Section 3, high performance implementation of PSO algorithms are presented. Experimental evaluations are discussed in Section 4. Finally, conclusions are presented in Section 5.

2. BACKGROUND

Background information about the construction of TPSO, APSO and ALC-PSO, as well as a brief description of graphics processing unit are presented in this section.

2. 1. Particle Swarm Optimization Particle swarm optimization algorithm was introduced by Eberhart and Kennedy [11-13]. In this algorithm each member of the swarm is called a particle and each swarm is called a group. The swarm is initialized with stochastic values and during movement, each particle remembers its best previous position and its best neighborhood's previous position which is represented by pBesti (pBesti1, pBesti2, … , pBestin) and gBest (gBest1, gBest2, … , gBestn) for particle i (i = 1, 2, ..., N), respectively. All particles exchange their good position with each other and according to this information, they set their position and velocity dynamically. In TPSO, particles update their velocity and position during each iteration using Equations (1) and (2), respectively, where for ith particle the velocity vector by $V_i(v_i^1, v_i^2, \dots, v_i^n)$ and the position vector is shown by $X_i(x_i^1, x_i^2, \dots, x_i^n)$. In addition, N is the number of particles in a population [14-18].

$$v_i^j = \omega v_i^j + c_1.r_1^j.\left(pBest_i^j - x_i^j\right) + c_2.r_2^j.\left(gBest^j - x_i^j\right) \quad (1)$$

$$x_i^j = x_i^j + v_i^j \quad (2)$$

where c_1 and c_2 are acceleration coefficients, ω is the inertia weight and j (j = 1, 2, ..., n) represents the jth dimension of the search space. r_1^j and r_2^j which maintain diversity of the population, are randomly distributed in [0,1]. We have divided this algorithm into three parts, initialization, checking, and updating.

2. 2. Adaptive Particle Swarm Optimization Adaptive Particle Swarm Optimization (APSO) is one of the improved PSO algorithms. In addition to overcome the problem of premature convergence, APSO can improve the search efficiency and convergence speed by controlling the inertia weight acceleration coefficient and other algorithmic parameters automatically at run time.

At first, it performs an Evolutionary State Estimation (ESE) to identify one of the following four defined evolutionary states, including exploration, exploitation, convergence and jumping out in each generation using some information about population distribution and particle fitness. Then, it applies an Elitist Learning Strategy (ELS) to the globally best particle in order to help jumping out of local optima regions when the evolutionary state is classified as convergence state. The effects of parameter adaptation show potential improvements that have been gained by this algorithm in comparison with TPSO [6].

2. 3. Particle Swarm Optimization with an Aging Leader and Challengers Particle Swarm Optimization with an Aging Leader and Challengers (ALC-PSO) is another improved version of TPSO. It has been developed by considering the effect of aging on the cultural diversity of a social animal colony. In nature, if the leader of a colony be too old, it does not have adequate leading power to challenge and claim the leadership, so it must be replaced by new individuals. According to aging mechanism, the gBest cannot be the leader necessarily, but a particle with adequate leading power is the leader. The ALC-PSO maintains the population diversity and overcomes the problem of getting trapped in local optima without weakening the fast-converging feature of TPSO [2].

2.CUDA, C programming guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide. (2014).

2. 4. Graphics Processing Unit Graphic Processing Unit (GPU) which has just been used to perform graphical process in recent years, is being used for non-graphics and general-purpose computing applications [3, 19]. Actually, due to GPU construction that involves multiple cores driven by very high memory bandwidth, its parallel computing mechanism and fast floating-point operation, it offers incredible resources for both graphics and non-graphics processing. GPU can support concurrent execution of tens of thousands of threads because of its massively parallel computing architecture [20]. So, it is more appropriate for data-parallel computations and is able to do more floating-point operations per second [3]. In order to exert the full power of a GPU, it is necessary to consider a good usage of the computing units and memory systems [21]. Compute Unified Device Architecture (CUDA) is a programming model and parallel computational platform which has been developed by NVIDIA to perform general-purpose computing on GPU conveniently. The programmability of GPU hardware has been dramatically increased by introduction of CUDA [20, 22-25].

2. 5. Related Work Many researchers have focused on performance improvement of PSO algorithms [3, 16, 26-36]. The standard PSO has been implemented on both GPU and CPU for four benchmark test functions. The maximum speedup was 11x [3]. The speedup of GPU based PSO with triggered mutation over CPU was 25x in [16]. In addition, asynchronous parallel PSO [26], using PC cluster system [27], implementing PSO based on the MapReduce parallel programming model [28] and using GPU [29] are such a parallel implementations which have been proposed in literature. Some of GPU based PSO algorithms are given in Table 1. To the best of our knowledge, there is not any high performance implementation of APSO and ALC-PSO algorithms in literature.

3. HIGH PERFORMANCE IMPLEMENTATION OF TRADITIONAL AND IMPROVED PSO

In order to implement the algorithms on GPU, we need to decompose tasks in order to determine which parts of the program can be executed independently. There are variant decomposition techniques such as recursive, input data, output data, exploratory, and speculative decompositions. We have achieved a good speedup using loop level parallelism, removing loops and executing the code of each loop on parallel threads. TPSO, APSO and ALC-PSO in parallel are implemented as follows. We consider NP and ND as the number of particles and the number of dimensions of the problem, respectively. The lower and upper bounds of the problem are defined as [Start_Range_Min, Start_Range_Max]. The most important arrays which maintain particles' information are defined as follows.

- x[NP*ND]: current position of the particles
- v[NP*ND]: current velocity of the particles
- pBests[NP*ND]: the best current position of the particles
- gBest[ND]: the best position of the swarm

The arrays have been considered one dimensional because we have to save them on global memory when we want to transfer them into GPU and only one dimensional array can be accepted by global memory.

In TPSO algorithm after initialization, updating velocity and position of particles and pBest are executed in parallel, while there is write after write data dependency among some tasks. Therefore, updating gBest must be run sequentially. Particles must be initialized randomly. So, first of all, we execute random generator function on CPU, and then transfer the arrays to GPU. In APSO algorithm, the tasks which are relevant to calculating the mean distance of each particle from other particles, velocity, position and pBest update can be executed in parallel, while the tasks which are relevant to gBest update, because of their data dependency,` cannot be run in parallel.

In ALC-PSO implementation instead of gBest, two arrays have been defined as follows.

- Leader[ND]: current leader of the swarm
- Challenger[ND]: this is a particle that is replaced with the current leader if the leader is too old.

Leader is the only leader of ALC-PSO that has an important impact on all of particles; hence its updating must be executed sequentially. In order to compare the best position of each particle with the leader and doing update, the array of pBests that has been updated on GPU must be transferred into CPU. In this algorithm, updating the velocity and position of each particle are performed in parallel.

4. EXPERIMENTAL EVALUATIONS

In this section, we present our experimental results which have been obtained on CPU and GPU platforms.

4. 1. Benchmark Functions and Implementation Environment For performance comparisons between CPU and GPU implementations, six classical benchmark test functions which are depicted in Table 2 have been selected. These test functions can be classified into two groups. The first three functions -F_1 - F_3- are unimodal functions and the next three -F_4 – F_6- are multimodal functions. Actually, these two kinds of test functions have been chosen to show that those algorithms can be used for both simple and complex functions.

TABLE 1. Variant GPU-based implementation of PSO algorithms

Algorithm	Description	Reference
Standard Particle Swarm Optimization (SPSO)	SPSOs have been implemented on both GPU and CPU to optimize four benchmark test function. Experimental results showed that GPU-SPSO can be 11x faster than CPU-SPSO.	[3]
Particle Swarm Optimization with Triggered Mutation(PSO-TM)	Speedup of GPU implementation of PSO-TM over CPU is 25x.	[16]
Particle swarm optimization (PSO)	An implementation of PSO algorithm in C-CUDA has been presented in order to reduce computational time. The best performance over the C implementation was 17x.	[30]
Multi-Swarm Particle Swarm Optimization (MSPSO)	A collaborative multi-swam PSO algorithm has been implemented on GPU and the results show that GPU based MSPSO can be 37x faster than its sequential implementation.	[31]
Parallel Dimension Particle Swarm Optimization (PDPSO)	Speedup of GPU based implementation of PDPSO was 85xover CPU basedl implementation.	[32]
PSO	A CUDA accelerated PSO has been used in order to reduce the computational time of the multidimensional knapsack problem. The attainable performance benefit has been evaluated when using a highly optimized GPU code instead of an efficient multi-core CPU implementation and 9.6x has been gained as the highest speedup.	[33]
PSO	In order to enhance the efficiency of the PSO algorithm, it has been implemented using the shared memory available in the GPU of CUDA platforms. In this implementation, each dimension of each particle has been mapped as a thread and multiple sub-swarms have been used. The results show the speedups up to 100x have been achieved compared to the serial implementation.	[34]
PSO and Distributed PSO (DPSO)	In order to get the maximum efficiency while solving large size maximal constraint satisfaction problems, PSO and Distributed PSO have been implemented in parallel using GPU architecture. Speedups of up to 3.79x have been gained.	[35]
PSO	Inherent parallelism of GPU has been utilized in order to accelerate the computing time of PSO algorithm and benefit of parallel computing mechanism supported by general purpose computing ability of GPU has been taken. The efficiency of the algorithm has been tasted on five different functions and the speedup of 30x have been gained.	[36]

TABLE 2. Benchmark test functions

Name	Test Functions	Domain
Sphere	$F_1(x) = \sum_{i=1}^{n} x_i^2$	[-100, 100]
Schewefel's P2.22	$F_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	[-10, 10]
Zakharov	$F_3(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5 i x_i)^2 + (\sum_{i=1}^{n} 0.5 i x_i)^4$	[-10, 10]
Rastrigin	$F_4(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)] + 10n$	[-5.12, 5.12]
Griewank	$F_5(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[-600, 600]
Michal-ewicz	$F_6(x) = \sum_{i=1}^{n} \sin(x_i) . (\sin\left(\frac{i x_i^2}{\pi}\right))^{2.m}$	[0, π]

In multimodal functions, optimization algorithms try to find different good solutions. In other words, multimodal functions are those with multiple local optima. Each of these functions has specific characteristics which can have impact on their behavior. For example, Rastrigin function is separable, asymmetrical and with huge number of local optima. Some of them have geometric or more floating point computations. But, CPU and GPU face with these characteristics in a different way. For instance, in comparison with sin and cos calculations, floating point calculations can be performed 6x faster on our GPU compared to CPU [32].

We executed all three algorithms under the same conditions. In other words, number of dimensions and threads are 5 and 64 in all implementations, respectively. We have executed GPU and CPU based programs several times independently.

In order to compare the GPU implementations with the fastest CPU implementations, we have executed CPU-TPSO, CPU-APSO and CPU-ALCPSO for 2000 particles on three different kinds of CPUs. The characteristics of these three CPUs are shown in Table 3. After comparing these CPUs with each other, CPU C and GPU GeForce GT 740M have been selected.

TABLE 3. The specifications of the CPUs platforms.

Name	Description
CPU A	AMD Sempron(TM) 145 Processor 2.80 GHz
CPU B	Intel(R) Core(TM) 2 Duo CPU T9300 2.5 GHz
CPU C	Intel Core i5-4200M 2.50GHz

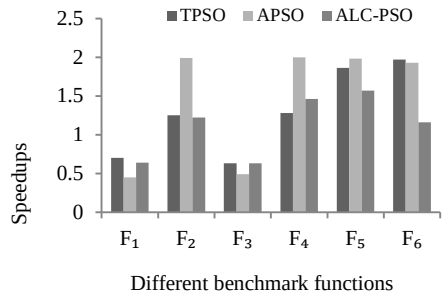

Figure 1. Speedup of CPU B over CPU A for three PSO algorithms using different bechmarks functions.

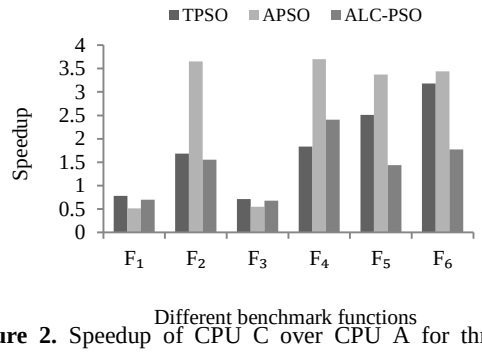

Figure 2. Speedup of CPU C over CPU A for three PSO algorithms using different benchmarks functions.

We have used the GPU GeForce GT 740M in order to evaluate the effect GPU threads on the performance of APSO and ALCPSO executing.

4. 2. Experimental Results on Different CPUs The PSO algorithms have different behavior on different kinds of processors. The speedups of CPU B and CPU C over CPU A are depicted in Figures 1 and 2, respectively. The CPU B yields speedup ranging from 0.45 to 2 and CPU C yields speedup ranging from 0.5 to 3.7. As can be seen, for most functions, the performance improvement of CPU C is more than the CPU B. Hence, CPU C was chosen for our performance comparison with GPU based implementations.

4. 3. Performance Comparison on CPU and GPU We have executed these benchmark functions on CPU and GPU platforms in two different ways based on the number of iterations, variable and fixed. In the first implementation, since we already knew the optimum value of the benchmark test functions, algorithms were executed until they reach the optimum value. In other words, the CPU and GPU based programs were stopped as soon as they reached the optimum value and acceptable solution. In the second implementation, algorithms were executed with predefined or fixed number of iterations such as 1000 and 4000.

4. 5. Speedup of GPU over CPU with Predefined Optimum Value The experimental results on CPU and GPU for TPSO, APSO and ALC-PSO algorithms with predefined optimum value are depicted in Table 4. The number of particles and dimensions are 1000 and 5, respectively. In the table, columns, CPU-Iter and GPU-Iter represent the number of iterations after that the programs were stopped and reached to the predefined optimum value. The second, fifth, and eighth columns represent the speedup of GPU-TPSO, GPU-APSO, GPU-ALCPSO over CPU-TPSO, CPU-APSO, and CPU-ALCPSO, respectively.

As Table 4 depicts, the GPU-TPSO yields speedup ranging from 1.57 to 10.9, while the GPU-APSO yields speedup ranging from 10.42 to 62.91. The GPU-ALCPSO yields speedup ranging from 1.05 to 27.63.

For most functions, GPU iteration is less than CPU iteration. For example, for Griewank function in TPSO algorithm in Table 4, the CPU-Iter and GPU-Iter are 165 and 35, respectively. This means that GPU can reach the predefined optimum value with less iteration. Since in PSO algorithms, particles are initialized randomly, if we let the programs to be run until achieving the optimum value, they stop in different number of iterations in each execution.

4. 6. Performance of GPU over CPU with Predefined Number of Iterations To make a fair performance comparison between CPU and GPU, we have implemented three algorithms using fixed number of iterations. Numbers of iterations have been considered 4000 for TPSO and ALC-PSO and 1000 for APSO algorithms.

Speedups of GPU-TPSO, GPU-APSO, and GPU-ALCPSO implementations over CPU-TPSO, CPU-APSO, and CPU-ALCPSO implementations are depicted in Figures 3, 4 and 5, respectively. The number of particles are selected from 1000 to 3000. As can be seen in these figures, with increasing the number of particles, the speedups are also increased. For example, in F1 function, the speedup of GPU-TPSO over CPU-TPSO is 9.3x and 14.5x for N = 1000 and N = 3000, respectively.

Speedups of GPU-APSO and GPU-ALCPSO are more than the speedups of GPU-TPSO algorithm. The maximum speedup in GPU-TPSO is 14.5 while the maximum speedups in GPU-APSO and GPU-ALCPSO are 152 and 31, respectively. There are more parallelism in APSO and ALC-PSO algorithms compared to TPSO. Especially, in APSO algorithm, there are many nested loops which have been exploited using loop- and data-level parallelism. In addition, speedup is increased by increasing the number of particles and number of iterations; even GPU-APSO can be 300x faster than CPU-APSO for 4000 particles.

4. 6. Evaluating the Effect of GPU Threads on the Performance of APSO and ALC-PSO Executing On the GPU based programs, choosing the appropriate number of threads based on the selected PSO algorithm, the features of employed GPU and population size can impact on the performance of parallel programs and their execution times. In fact, the occupancy of GPU blocks -number of threadsper block- can impact on the performance of program execution. In order to evaluate the impact of threads, GPU-APSO and GPU-ALCPSO have been executed using 500 and 1000 particles, respectively. The speedup of GPU-APSO over CPU-APSO and the speedup of GPU-ALCPSO over CPU-ALCPSO using 64,128, 256, 512 and 1024 threads per block have been depicted in Figures 6 and 7 for 500 and 1000 particles, respectively. As can be seen, the execution times of GPU based program is reduced with increasing the number of particles. Although in all cases of using different threads, the performance of GPU based programs is better than the performance of its CPU based programs, GPU shows its best performance when 100% of block capacity is used. In the other words, GPU based programs can be executed in the smallest time when the block occupancy is 100%. The occupancy of each multiprocessor which can impact on the programs performance and the behavior of programs in dealing with different functions can be calculated using Equation (3).

$$\text{The occupancy of each multiprocessors} = \frac{\text{The number of allocated warps per multiprocessors}}{\text{Maximum number of warps which are supportable by each multiprocessor}} \quad (3)$$

Since each multiprocessor contains a set of registers, the number of employed registers by the program, computational capability and physical limitation of GPU can have impact on the performance of programs in dealing with different functions.The occupancy of each multiprocessor according to the number of allocated registers per threads for GeForce GT 740M is shown in Table 5. When the number of threads per block is 64, the number of warps per block is 2.

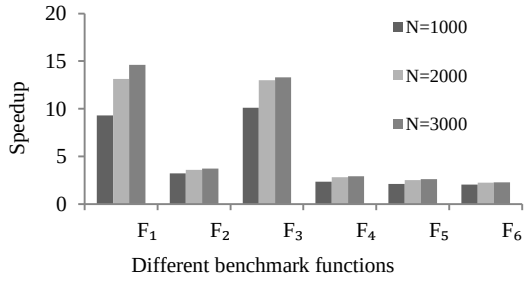

Figure 3. Speedup of GPU-TPSO implementation over CPU-TPSO implementation for different benchmarks with different number of particles.

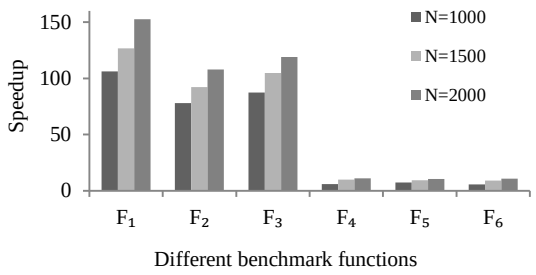

Figure 4. Speedup of GPU-APSO implementation over CPU-APSO implementation for different benchmarks with different number of particles.

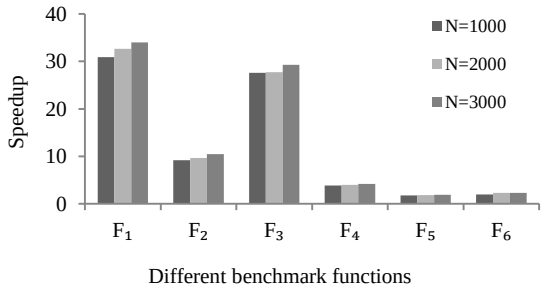

Figure 5. Speedup of GPU-ALCPSO implementation over CPU-ALCPSO implementation for different benchmarks with different number of particles.

Since the maximum number of active blocks per multiprocessor can be 16, the number of active warps per multiprocessor will be 32, while each multiprocessor in our employed GPU can support 64 warps in parallel. Hence, the number of active blocks which can be supported by each multiprocessor can cause performance limitation in this case. Considering Equation (3), the occupancy of multiprocessors per block for 64 threads is 50%.

When the number of threads per block is 128, the number of warps per block is 4. Since the maximum number of active blocks per multiprocessor can be 16, the number of active warps per multiprocessor will be 64. So, the maximum number of blocks can cause performance limitation. Although the occupancy of each multiprocessor per block is 100% for 128 threads, GPU hardware limits performance in this case. When the number of threads per block is 256, 512 and 1024, the number of warps per block is 8, 16 and 32, respectively. In these cases, the number of active warps per multiprocessor can cause performance limitation. The number of active warps per multiprocessor for 256, 512 and 1024 threads is 64 but the number of active blocks per multiprocessor is 8, 4 and 2, respectively.

In this case, each multiprocessor can support up to 16 blocks, and considering Equation (3), the occupancy of multiprocessors is 100% for 256, 512 and 1024 threads.

TABLE 4. The execution resultsof TPSO, APSO and ALC-PSO algorithms on CPU and GPU with predifined optimum value.

	TPSO			APSO			ALC-PSO		
Name	**Speedup**	**CPU-Iter**	**GPU-Iter**	**Speedup**	**CPU-Iter**	**GPU-Iter**	**Speedup**	**CPU-Iter**	**GPU-Iter**
Sphere	3.7	31	24	10.42	41	22	3.1	33	28
Schwefel	10.85	28	37	11.36	39	19	2.9	37	31
Zakharov	4.88	26	15	22.11	50	12	4.71	34	18
Rastrigin	1.57	65	81	23.44	35	27	1.05	80	137
Griewank	10.95	165	35	62.91	310	27	27.63	603	39
Michalewicz	2.88	30	11	21.79	98	21	3.03	33	18

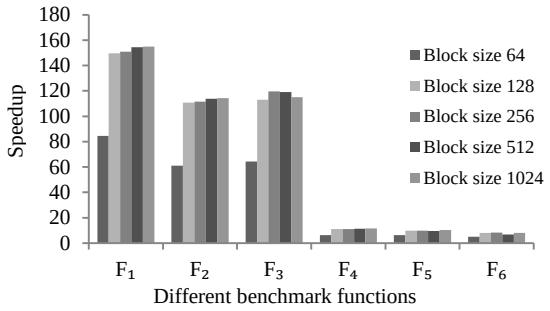

Figure 6. Speedup of GPU-APSO over CPU-APSO using 500 particles, six benchmark functions and predefined number of iterations for different block sizes.

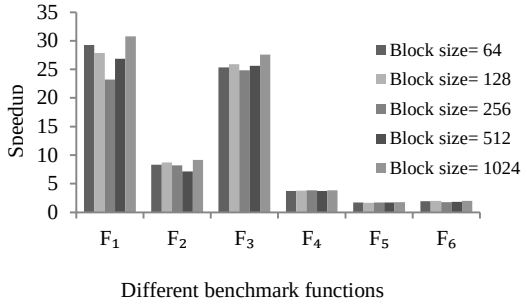

Figure 7. Speedup of GPU-ALCPSO over CPU-ALCPSO using 1000 particles, six benchmark functions and predefined number of iterations for different block sizes.

TABLE 5. The multiprocessors features while the different benchmark functions are running on GeForce GT 740 M

The occupancy of each multiprocessor	The number of active blocks per multiprocessor	The number of active warps per multiprocessor	The number of active threads per multiprocessor	The number of registers per thread	The number of threads per block
50%	16	32	1024	9, 11,16, 21, 22	64
100%	16	64	2048	9, 11, 16, 21, 22	128
100%	8	64	2048	9,11,16, 21,22	256
100%	4	64	2048	9,11,16, 21,22	512
100%	2	64	2048	9, 11,16, 21,22	1024

Even allocating 1024 threads to each block and 22 registers to each thread, totally 45056 registers per multiprocessor is required. Hence, there is no limitation in this GPU in term of number of registers and GPU behavior for same number of threads but different number of registers which are required for each thread, is almost the same.

5. CONCLUSIONS

Particle Swarm Optimization (PSO) algorithm, and its improved PSO algorithms such as Adaptive PSO (APSO) and PSO with an Aging Leader and Challengers (ALC-PSO) are used for optimizations problems. In comparison with the traditional PSO, the APSO and ALC-PSO algorithms improve convergence speed and avoid the problem of premature convergence. Similar to most of the evolutionary algorithms, these algorithms are population-based iterative and computationally intensive. The main reason is that the optimizing process of these algorithms requires a large number of fitness evaluations which runs sequentially on CPU. We have improved the performance of the mentioned algorithms on GPU platform in two different ways, variable and fixed number of iterations. In

variable number of iteration, we set predefined optimum value for all benchmark test functions, and maximum speedup were 10.9, 62.91, and 27.63 for TPSO, APSO, and ALC-PSO algorithms, respectively. In fixed number of iteration, 4000 for TPSO and ALC-PSO and 1000 for APSO algorithms were used. The maximum speedup in GPU-TPSO is 14.5, while the maximum speedups in GPU-APSO and GPU-ALCPSO are 152 and 31, respectively. Although the APSO and ALC-PSO improve the performance of TPSO algorithm in terms of convergence speed, global optimality and solution accuracy, they are more computational intensive than TPSO algorithm. Speedups of GPU-APSO and GPU-ALCPSO are more than the speedups of GPU-TPSO algorithm in all implementations. The largest speedup yields for APSO algorithm. There are much more nested-loops in APSO algorithm and it is exploited using loop- and data-level parallelism in parallel implementation. With increasing the number of particles and iterations, the speedup is also increased. In addition, the number of GPU threads can have impact on the performance of parallel implementation. Our experimental results show that the ideal occupancy -number of threads per block- for APSO and ALC-PSO is almost 100% of the total threads of the block. Actually, when we do not have any limitation in terms of number of registers, full block occupancy can be appropriate in order to reduce GPU based programs execution time.

6. REFERENCES

1. Rao, S.S. and Rao, S., "Engineering optimization: Theory and practice, John Wiley & Sons, (2009).
2. Altinoz, O.T. and Yilmaz, A.E., "Particle swarm optimization with parameter dependency walls and its sample application to the microstrip-like interconnect line design", ***AEU-International Journal of Electronics and Communications***, Vol. 66, No. 2, (2012), 107-114.
3. Zhou, Y. and Tan, Y., "GPU-based parallel particle swarm optimization", in IEEE Congress on Evolutionary Computation, (2009), 1493-1500.
4. Ciuprina, G., Ioan, D. and Munteanu, I., "Use of intelligent-particle swarm optimization in electromagnetics", ***IEEE Transactions on Magnetics***, Vol. 38, No. 2, (2002), 1037-1040.
5. Liang, J.J., Qin, A.K., Suganthan, P.N. and Baskar, S., "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions", ***IEEE Transactions on Evolutionary Computation***, Vol. 10, No. 3, (2006), 281-295.
6. Zhan, Z.-H., Zhang, J., Li, Y. and Chung, H.S.-H., "Adaptive particle swarm optimization", ***IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)***, Vol. 39, No. 6, (2009), 1362-1381.
7. Ho, S.-Y., Lin, H.-S., Liauh, W.-H. and Ho, S.-J., "OPSO: Orthogonal particle swarm optimization and its application to task assignment problems", ***IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans***, Vol. 38, No. 2, (2008), 288-298.
8. Liu, B., Wang, L. and Jin, Y.-H., "An effective PSO-based memetic algorithm for flow shop scheduling", ***IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)***, Vol. 37, No. 1, (2007), 18-27.
9. Chen, W.-N., Zhang, J., Lin, Y., Chen, N., Zhan, Z.-H., Chung, H.S.-H., Li, Y. and Shi, Y.-H., "Particle swarm optimization with an aging leader and challengers", ***IEEE Transactions on Evolutionary Computation***, Vol. 17, No. 2, (2013), 241-258.
10. Santos, A., Teixeira, J.M., Farias, T., Teichrieb, V. and Kelner, J., "Understanding the efficiency of kd-tree ray-traversal techniques over a gpgpu architecture", ***International Journal of Parallel Programming***, Vol. 40, No. 3, (2012), 331-352.
11. Eberhart, R.C. and Kennedy, J., "A new optimizer using particle swarm theory", in Proceedings of the sixth international symposium on micro machine and human science, New York, NY. Vol. 1, (1995), 39-43.
12. Eberhart, R. C., Kennedy, J., "Particle swarm optimization", In Proceedings of IEEE International Conferenceon Neural Networks, (1995); 1942–1948.
13. Kennedy, J. and Eberhart, R.C., "A discrete binary version of the particle swarm algorithm", in Systems, Man, and Cybernetics, in IEEE International Conference on Computational Cybernetics and Simulation., 1997, Vol. 5, (1997), 4104-4108.
14. Sadri, J. and Suen, C.Y., "A genetic binary particle swarm optimization model", in IEEE International Conference on Evolutionary Computation, (2006), 656-663.
15. Bratton, D. and Kennedy, J., "Defining a standard for particle swarm optimization", in IEEE swarm intelligence symposium, (2007), 120-127.
16. Zhou, Y. and Tan, Y., "Particle swarm optimization with triggered mutation and its implementation based on GPU", in Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM, (2010), 1-8.
17. Liao, C.-Y., Lee, W.-P., Chen, X. and Chiang, C.-W., "Dynamic and adjustable particle swarm optimization", in Proceedings of the 8th WSEAS International Conference on Evolutionary Computing, Citeseer, (2007), 301-306.
18. Mendes, R., Kennedy, J. and Neves, J., "The fully informed particle swarm: Simpler, maybe better", ***IEEE Transactions on Evolutionary Computation***, Vol. 8, No. 3, (2004), 204-210.
19. Kromer, P., Platos, J. and Snasel, V., "Nature-inspired meta-heuristics on modern GPUs: State of the art and brief survey of selected algorithms", ***International Journal of Parallel Programming***, Vol. 42, No. 5, (2014), 681-709.
20. Cao, Y., Patnaik, D., Ponce, S., Archuleta, J., Butler, P., Feng, W.-c. and Ramakrishnan, N., "Parallel mining of neuronal spike streams on graphics processing units", ***International Journal of Parallel Programming***, Vol. 40, No. 6, (2012), 605-632.
21. Shen, X., Liu, Y., Zhang, E.Z. and Bhamidipati, P., "An infrastructure for tackling input-sensitivity of GPU program optimizations", ***International Journal of Parallel Programming***, Vol. 41, No. 6, (2013), 855-869.
22. Sun, E. and Kaeli, D., "Aggressive value prediction on a GPU", ***International Journal of Parallel Programming***, Vol. 42, No. 1, (2014), 30-48.
23. Lee, C., Ro, W.W. and Gaudiot, J.-L., "Boosting CUDA applications with CPU–GPU hybrid computing", ***International Journal of Parallel Programming***, Vol. 42, No. 2, (2014), 384-404.
24. Andion, J.M., Arenaz, M., Bodin, F., Rodriguez, G. and Tourino, J., "Locality-aware automatic parallelization for GPGPU with openhmpp directives", ***International Journal of Parallel Programming***, Vol. 44, No. 3, (2016), 620-643.
25. Martinez-Angeles, C.A., Wu, H., Dutra, I., Costa, V.S. and Buenabad-Chavez, J., "Relational learning with GPUs: Accelerating rule coverage", ***International Journal of Parallel Programming***, Vol. 44, No. 3, (2016), 663-685.

26. Ziyabari, S.H.S. and Shahbahrami, A., "High performance implementation of apso algorithm using gpu platform", in International Symposium onArtificial Intelligence and Signal Processing (AISP), IEEE, (2015), 196-200.
27. Venter, G. and Sobieszczanski-Sobieski, J., "Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations", ***Journal of Aerospace Computing, Information, and Communication***, Vol. 3, No. 3, (2006), 123-137.
28. Kim, J.-Y., Jeong, H.-M., Lee, H.-S. and Park, J.-H., "PC cluster based parallel pso algorithm for optimal power flow", in International Conference on Intelligent Systems Applications to Power Systems, ISAP., IEEE, (2007), 1-6.
29. McNabb, A.W., Monson, C.K. and Seppi, K.D., "Parallel PSO using mapreduce", in IEEE Congress on Evolutionary Computation, (2007), 7-14.
30. Veronese, L.D.P. and Krohling , R.A., " Swarm's flight: Accelerating the particles using C-CUDA", in IEEE Congress on Evolutionary Computation, (2009), 3264-3270.
31. Solomon, S., Thulasiraman, P. and Thulasiram, R., "Collaborative multi-swarm PSO for task matching using graphics processing units", in Proceedings of the 13th annual conference on Genetic and evolutionary computation, ACM, (2011), 1563-1570.
32. Calazan, R.M., Nedjah, N. and de Macedo Mourelle, L., "Parallel GPU-based implementation of high dimension particle swarm optimizations", in IEEE Fourth Latin American Symposium on Circuits and Systems (LASCAS), (2013), 1-4.
33. Zan, D. and Jaros, J., "Solving the multidimensional knapsack problem using a cuda accelerated PSO", in IEEE Congress on Evolutionary Computation (CEC), (2014), 2933-2939.
34. Silva, E.H. and Bastos Filho, C.J., "PSO efficient implementation on GPUs using low latency memory", ***IEEE Latin America Transactions***, Vol. 13, No. 5, (2015), 1619-1624.
35. Dali, N. and Bouamama, S., "Parallel particle swarm optimization approaches on graphical processing unit for constraint reasoning: Case of max-CPSs", ***Procedia Computer Science***, Vol. 60, (2015), 1070-1080.
36. Kaur, J., Singh, S. and Singh, S., "Parallel implementation of PSO algorithm using GPGPU", in Second International Conference on Computational Intelligence & Communication Technology (CICT), IEEE, (2016), 155-159.

Parallel Implementation of Particle Swarm Optimization Variants Using Graphics Processing Unit Platform

S. Jam, A. Shahbahrami, S. H. S. Ziyabari

Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran

P A P E R I N F O

Paper history:
Received 23 June 2016
Received in revised form 20 September 2016
Accepted 11 November 2016

Keywords:
Particle Swarm Optimization
Adaptive Particle Swarm Optimization
Particle Swarm Optimization with an Aging Leader and Challengers
Graphics Processing Unit

چکیده

انواع مختلفی از الگوریتم‌های بهینه‌سازی اجتماع ذرات (PSO)، از جمله الگوریتم تطبیقی بهینه‌سازی اجتماع ذرات (APSO) و الگوریتم بهینه‌سازی اجتماع ذرات با رهبر سالخورده و رقبا (ALC-PSO) وجود دارد. اگر چه این الگوریتم‌ها در زمینه‌ی یافتن بهترین پاسخ و تسریع سرعت همگرایی موجب بهبود عملکرد الگوریتم PSO می‌شوند، اما دارای حجم محاسباتی بالایی هستند. هدف اصلی این مقاله، پیاده‌سازی کارآمد الگوریتم سنتی PSO (TPSO)، APSO و ALC-PSO با استفاده از فناوری CUDA است. به منظور ارزیابی و بهبود عملکرد این سه الگوریتم و کاهش زمان اجرای آنها، این الگوریتم‌ها را بر روی هر دو پلتفرم CPU و GPU پیاده سازی کرده‌ایم. برای سه الگوریتم TPSO، APSOو ALC-PSO ، به ترتیب به تسریعی برابر با ۱۴،۵، ۳۱ و ۱۵۲ دست یافته‌ایم. به علاوه، به منظور یافتن تعداد نخ‌های مناسب به ازای هر بلاک، برای دو الگوریتم APSO و ALC-PSO، ما از تعداد نخ‌های مختلفی در پیاده‌سازی‌های انجام شده، استفاده کرده‌ایم. نتایج به دست آمده نشان می‌دهد که انتخاب بهترین تعداد نخ‌ها به ازای هر بلاک، به تعداد متغیرها و ثابت‌های موجود در هر الگوریتم و تعداد رجیسترها به ازای هر چندپردازنده بستگی دارد.

***doi**: 10.5829/idosi.ije.2017.30.01a.07*