High-accuracy Optimization by Parallel Iterative Discrete Approximation and GPU Cluster Computing

Di Zhao[1,2]
[1]Center for Cognitive and Brain Science, The Ohio State University
[2]College of Medicine, The Ohio State University
Columbus, OH 43210
Email: zhao.1029@osu.edu
Web: https://cog.osu.edu/people/zhao

***Abstract*—High-accuracy optimization is the key component of time-sensitive applications in computer sciences such as machine learning, and we develop single-GPU Iterative Discrete Approximation Monte Carlo Optimization (IDA-MCS) and multi-GPU IDA-MCS in our previous research. However, because of the memory capability constrain of GPUs in a workstation, single-GPU IDA-MCS and multi-GPU IDA-MCS may be in low performance or even functionless for optimization problems with complicated shapes such as large number of peaks. In this paper, by the novel idea of parallelizing Iterative Discrete Approximation with CUDA-MPI programming, we develop the GPU cluster version (GPU-cluster) of IDA-MCS with two different parallelization strategies: Domain Decomposition and Local Search, under the style of Single Instruction Multiple Data by CUDA 5.5 and MPICH2, and we exhibit the performance of GPU-cluster IDA-MCS by optimizing complicated cost functions. Computational results show that, by the same number of iterations, for the cost function with millions of peaks, the accuracy of GPU-cluster IDA-MCS is approximately thousands of times higher than that of the conventional method Monte Carlo Search. Computational results also show that, the optimization accuracy from Domain Decomposition IDA-MCS is much higher than that of Local Search IDA-MCS.**

***Index Terms*—GPU Cluster Computing; CUDA-MPI Programming; Iterative Discrete Approximation; High-accuracy Optimization; Domain Decomposition; Local Search**

I. INTRODUCTION

Numerical optimization is computer based algorithms to calculate the value of optimum, which typically solves one-dimensional (1D) or high-dimensional cost function with the form such as the two-dimensional (2D) cost function:

$$\max_{\Omega} f(x, y), \tag{1}$$

where Ω is search space, a finite area for optimization, and the two variables $(x, y) \in \Omega$. The GPU based approaches with the style of Single Instruction Multiple Data (SIMD) to solve high-dimensional optimization problems include Particle Swarm Optimization, Brute-force Search, Monte Carlo Search, *etc*.

Particle Swarm Optimization is a method of optimization with scanning the search space by a group of candidate solutions named particle, and these particles are suitable for parallelization. Particle Swarm Optimization is well parallelized by GPU computing [1-6] with application to computer sciences [7-14], finance [15, 16], physics [17], biology [18], *etc*.

Brute-force Search is an optimization method to exhaustively search all possibility in the search space on uniform grids, and the accuracy of optimum almost depends on the coverage of search space. Brute-force Search successfully optimize the problems from computer sciences [19-26], finance [27, 28], physics [29], chemistry [30], biology [31-35], *etc*.

Brute-force Search is suitable for solving the low-dimension problem, and this algorithm has nice property of parallelization, which leads to high-performance in GPU computing. However, for the high-dimensional cost function, the computational cost of Brute-force Search dramatically increases because of *curse of dimensionality*. To solve this problem, methods such as Monte Carlo Search are developed.

Different from uniform grid in Brute-force Search, Monte Carlo Search calculates optimization by random numbers of different distribution, especially for high-dimensional optimization. By Monte Carlo Search, the cost function equation (1) can be calculated by sampling a set of random numbers with total number of I:

$$\max_{\Omega} f(x, y) = \max_{i \in I} f(x_i, y_i), \tag{2}$$

where $0 < i < I$ and $(x, y) \in \Omega$. If the distribution of the cost function is not estimated, the distribution for sampling random numbers is usually uniform distribution, and proper estimation of the distribution of the cost function increases the efficiency of mixing. As a conventional method, Monte Carlo Search is successfully applied to optimize the problems from fields such as computer sciences [8, 36], biology and medical healthcare [2, 7, 37], physics [38], economics [3, 39], *etc*.

doi:10.4304/jsw.9.9.2366-2377

Monte Carlo Search covers the whole search space by sampling large number of random points. However, it is computational challenging for the conventional Monte Carlo Search to high-accuracy optimize a cost function $f(x, y)$ with a complicated shape such as thousands to millions of peaks in real-time.

In this paper, by the novel idea of parallelizing Iterative Discrete Approximation by CUDA-MPI programming, we develop GPU-cluster versions of Iterative Discrete Approximation Monte Carlo Optimization (IDA-MCS) with two different parallelization strategies: Domain Decomposition and Local Search, with the style of SIMD by CUDA 5.5 and MPICH2, and we exhibit the performance of GPU-cluster IDA-MCS by optimizing complicated functions with millions of peaks.

Additionally, for the convenience of later description, we clearly define the terms of single-GPU, multi-GPU and GPU-cluster. In this paper, single-GPU means a GPU in a GPU workstation, multi-GPU means multiple GPUs in a GPU workstation, and GPU-cluster means multiple GPUs in a GPU cluster of multiple nodes.

II. Methods

A. 2D Iterative Discrete Approximation

For a continuous 2D cost function $f(x, y)$, how to produce a set of random numbers whose distribution obeys the given 2D cost function $f(x, y)$? Discrete Approximation is a method to answer this question, and Discrete Approximation is a method to generate the approximation of 2D discrete function $f(x, y)$ by a set of random samples, and the distribution of random numbers obeys the 2D function $f(x, y)$ directly or indirectly, and the total number of random samples is usually preset. Setting total number of random samples results in the style of SIMD and then advantages in high-efficient applications, and this is the first reason why the high-accuracy of GPU-cluster IDA-MCS comes from.

To discretely approximate a 2D function $f(x, y)$, for a given set of random numbers $A = (x_i, y_i)$, the set $f(A)$ returns implicit information about the 2D function. Applying the weighting function w to the set $f(A)$, the function

$$w(f(A)) \tag{3}$$

returns amplified implicit information about the cost function. To transform the implicit information to the explicit form, a Monte Carlo Simulation is applied to $w(f(A))$, and with current design a new set B is generated by

$$B \sim cumsum(w(f(A))),$$

where *cumsum* means cumulative sum, and ~ means Monte Carlo Simulation. By some transform function t, a set A_{new} is produced by

$$A_{new} = t(B),$$

where A_{new} is Discrete Approximation of the cost function $f(x, y)$.

In some applications, since the shape of the function $f(x, y)$ is sometimes too complicated, Discrete Approximation is not powerful enough to approximate the cost function under such circumstance. To solve this problem, we bring the novel idea of introducing iteration into Discrete Approximation in our previous research, which leads to Iterative Discrete Approximation. After iterations, the points become more and more concentrating on the peak area, which significantly improves the accuracy of optimizing the cost function $f(x, y)$.

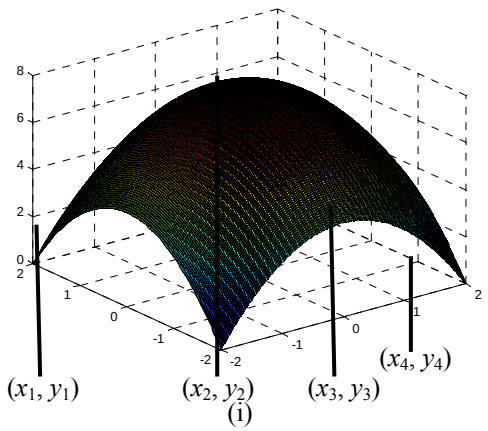
Let us explain how Iterative Discrete Approximation works by an example in the language of GPU computing, as shown in Figure 1. For a given 2D optimization function $f(x, y)$ with the search space, suppose we draw four samples (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, y_4) from a cost function in Figure 1 (i), and the cost function is:

$$f(x, y) = 8 - x^2 - y^2$$
$$-2 \le x \le 2, -2 \le y \le 2, \tag{4}$$

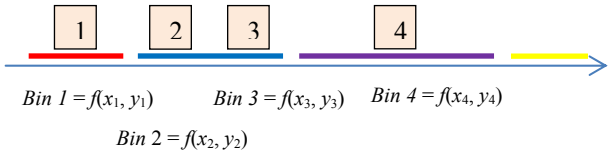
pass these four values through the cost function $f(x, y)$ with the style of SIMD, we obtain four values: $f(x_1, y_1)$, $f(x_2, y_2)$, $f(x_3, y_3)$ and $f(x_4, y_4)$ in Figure 1 (i), and the values of $f(x_1, y_1)$, $f(x_2, y_2)$, $f(x_3, y_3)$ and $f(x_4, y_4)$ contain the information of the shape and the locations of the cost function $f(x, y)$.

To take advantages of the implicit information, Iterative Discrete Approximation constructs bins. In Figure 1 (ii), there are four bins are constructed since there are four available estimation of the cost function, and the length of each bin is decided by the values of $f(x_1, y_1)$, $f(x_2, y_2)$, $f(x_3, y_3)$ and $f(x_4, y_4)$. Since the values of $f(x_1, y_1)$, $f(x_2, y_2)$, $f(x_3, y_3)$ and $f(x_4, y_4)$ are decided by the shape of the peak and the weighting function, the length of the bins are decided by the shape of the cost function indirectly.

Then a procedure of Monte Carlo Simulation is followed. Generate a set of random numbers of standard uniform distribution, for example four random numbers, and "throw" these numbers to these bins. Since the lengths of these bins are different, the possibility of "receiving" a random number is different, which is shown in Figure 1 (ii). After all iterations, the difference of bin lengths make the random numbers gradually gather in the peak area.



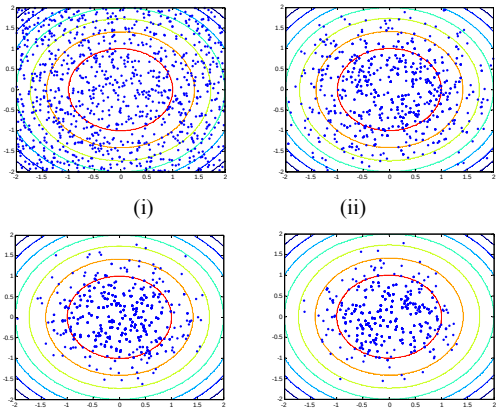
(i)



(ii)

Figure 1. Example of Iterative Discrete Approximation: (i) the Sampling Points from the Cost Function and (ii) the Constructed Bins by the Sampling Points

In real implementation of Iterative Discrete Approximation, the number of bins is typically large, although there may be no individual bin receiving large number of samples, bins locating the peak area receive most random numbers. This concentration brings tremendous advantages of operating $f(x, y)$ such as optimization and integration.



(i) (ii)

(iii) (iv)

Figure 2. Demonstration of Iterative Discrete Approximation with (i) the Input, (ii) the Output with $\beta = 2$, (iii) the Output with $\beta = 5$ and (iv) the Output with $\beta = 10$

To illustrate the performance of Iterative Discrete Approximation for 2D cost functions, we discretely approximate the cost function equation (4) with 1000 points in Figure 2, and the weighting function is:

$$f_w = \beta^{f(x,y)}$$

with different values of base $\beta = 2$, $\beta = 5$ and $\beta = 10$. The contour of equation (4) in Figure 1 (i) is also plotted in Figure 2. From Figure 2 we can clearly see, while the input points locate uniformly in Figure 2 (i), these points concentrate on the peak in Figure 2 (ii), and these points concentrate more on the peak in Figure 2 (iii) and Figure 2 (iv).

B. Single-GPU IDA-MCS for 1D Cost Function

We developed single-GPU IDA-MCS for the 1D cost function in our previous research, and single-GPU IDA-MCS aims at optimizing a function with relatively small-scale problems such as hundreds of peaks for high accuracy and efficiency, and we make a short introduction of single-GPU IDA-MCS for 1D cost function in this section.

Monte Carlo Search is the conventional method to optimize a cost function with the style of SIMD, and Monte Carlo Search can be summaries in Algorithm 1:

Algorithm 1. Monte Carlo Search for 1D Cost Function

- Generate a sample set A_1 with the bandwidth N, pass the set A_1 through $f(x, y)$ to the set B_1;
- Pass the set B_1 through the cost function $f(x, y)$ to the set C_1;
- In single GPU, calculate the optimum from the set C_1;

In Algorithm 1, we name the value of N as bandwidth, because the number of points in all sets keeps the value N to satisfy the requirements of SIMD. In our previous research, Algorithm 1 is accelerated by Discrete Approximation. To further increase the performance of Discrete Approximation, Iterative Discrete Approximation is developed, and finally arrives at single-GPU IDA-MCS of Algorithm 2:

Algorithm 2. Single-GPU IDA-MCS for 1D Cost Function

- Iteration 1: generate a set A_2^1 with the number of N elements, pass the set A_2^1 through the cost function $f(x, y)$ to the set B_2^1;
- Iteration k: discretely approximate the set B_2^k to the set C_2^k, pass the set C_2^k through equation (3) to the set D_2^k, and substitute D_2^k back to B_2^k for next iteration k+1;
- In single GPU, calculate the optimum from the set D_2;

In single-GPU IDA-MCS for 1D cost function, because of the limited memory capacity in single GPU, the value of N cannot be too big. For example, to approximate a discrete distribution with 10^4 points by 10^4 random numbers, CUDA array arrives at the size of 10^8, which usually reaches the maximum array size of GPUs. To solve this problem, multi-GPU IDA-MCS for 1D cost function is developed, which takes advantages of parallel versions of Iterative Discrete Approximation and multi-GPU programming to improve the accuracy of IDA-MCS.

C. Multi-GPU IDA-MCS for 1D Cost Function

How to make Iterative Discrete Approximation work with multi-GPU for 1D cost function with thousands of peaks? In our previous research, we apply two approaches to parallelize IDA-MCS by multi-GPU in a workstation: Domain Decomposition and Local Search for 1D optimization function.

Let's firstly make a short description of Domain Decomposition multi-GPU IDA-MCS for 1D function. By Domain Decomposition multi-GPU IDA-MCS, the search space is divided into domains with the total number of M, the total number of GPU is also M, and each GPU m is responsible for a domain m, where m is the counter for both GPU and the domain. In every iteration of IDA-MCS for 1D cost function, the computation of Iterative Discrete Approximation is equally distributed to multiple GPUs by Domain Decomposition, and the details are described in Algorithm 3:

Algorithm 3. Domain Decomposition multi-GPU IDA-MCS for 1D Cost Function

- In domain m, generate a set A_{3m}^{1} in GPU m of the multi-GPU workstation;
- In domain m, pass the set A_{3m}^{1} through $f(x, y)$ to the set B_{3m}^{1};
- In domain m and iteration k: discrete approximate the set B_{3m}^{k} to the set C_{3m}^{k};
- In domain m and iteration k: pass the set C_{3m}^{k} through equation (3) to the set D_{3m}^{k};
- In domain m and iteration k: substitute D_{3m}^{k} back to B_{3m}^{k} for next iteration k+1;
- Calculate the local optima from the set D_{3m}, finally calculate the global optimum from the local optima;

For 1D cost function, Local Search is the other strategy to parallelize IDA-MCS by multi-GPU in a workstation. While Domain Decomposition IDA-MCS separates the whole 1D search space across multiple GPUs, Local Search IDA-MCS runs multiple copies of Iterative Discrete Approximation simultaneously to calculate local optima, and the global optimum is calculated from these local optima. Each Local Search is called a "particle". The details of Local Search IDA-MCS for 1D cost function are described in Algorithm 4:

Algorithm 4. Local Search Multi-GPU IDA-MCS for 1D Cost Function

- In particle m, generate a set A_{4m}^{1} in GPU m of multi-GPU workstation;
- In particle m, pass the set A_{4m}^{1} through $f(x, y)$ to generate the set B_{4m}^{1};
- In particle m and iteration k, discretely approximate the set B_{4m}^{k} to the set C_{4m}^{k}, pass the set C_{4m}^{k} through equation (3) to the set D_{4m}^{k}, substitute D_{4m}^{k} back to B_{4m}^{k} for next iteration k+1;
- In particle m, calculate the local optima from the set D_{4m}, calculate the global optimum from the local optima;

Domain Decomposition and Local Search for multi-GPU workstation in this section are for 1D search space, and the idea of Domain Decomposition and Local Search for high-dimensional cost function is relatively the same, but different implementation.

D. Parallelization Strategy for 2D Iterative Discrete Approximation

From this section, based on previous research, Iterative Discrete Approximation is applied to 2D optimization problem with millions of peaks on GPU cluster, and the parallel algorithms are designed on these two characteristics. Two parallelization strategies for GPU cluster, Domain Decomposition and Local Search, are developed. Therefore, we develop two strategies to parallelize Iterative Discrete Approximation to GPU clusters: Domain Decomposition.

Let's firstly discuss Domain Decomposition for 2D Iterative Discrete Approximation. In the M-by-N GPU cluster system, therefore each GPU is counted as GPU m in the node n. Based on this topology, the search space is divided into M domains, each node is responsible for a domain; each domain is further divided into N subdomains, and each GPU is responsible for a subdomain. All subdomain is continuously counted as l, and they can be exchanged by:

$$l = n \times M + m, \tag{5}$$

where $0 \le l \le M \times N$.

Domain Decomposition IDA-MCS to 2D search space can by illustrated by 4-by-4 GPU system. Since the total number of GPUs in 4-by-4 system is 16, by Domain Decomposition with a grid size of 4-by-4 decomposition, the search space is equally divided into 16 subdomains to keep the load balance of the GPU cluster, and each GPU is responsible for a subdomain. In each subdomain, the set of random numbers is independently generated, the distribution is discretely approximated with no communication among the subdomains.

Local Search is the other strategy to parallelize Iterative Discrete Approximation for 2D cost function on GPU clusters. By Local Search, the search space is not divided, but multiple Iterative Discrete Approximation work independently in the search space to produce local optima, then calculate the global optimum. Each GPU is responsible for a particle l.

Same to Domain Decomposition, supposing that we solve the problem by a GPU cluster with four nodes and four GPUs per node, and total 16 random sets are sampled. Each particle works with a random set, and each particle covers the whole search space. Each particle produces a local optimum, and the global optimum is calculated from these local optima.

E. Domain Decomposition GPU-cluster IDA-MCS for 2D Cost Function

Based on the parallel strategy Domain Decomposition discussed in the previous section, equation (2) can be rewritten as the following:

$$\max_{\Omega} f(x,y) = \max_{\Omega_1+\Omega_2+\cdots+\Omega_l} f(x,y) \tag{6.1}$$

$$= \max\left(\max_{\Omega_1} f(x,y), \max_{\Omega_2} f(x,y), \cdots, \max_{\Omega_l} f(x,y) \right) \tag{6.2}$$

In equation (6.1), the computation of each subdomain $\max_{\Omega_l} f(x,y)$ covers the whole search space Ω. In equation (6.2), the search space Ω is divided into l subdomains, the computation of each subdomain $\max_{\Omega_l} f(x,y)$ can be distributed to each GPU, and the computation is done simultaneously by CUDA kernel concurrency and MPI. Comparing between equation (6.1) and equation (6.2), because of reduction of search space

and GPU parallel computing, equation (6.2) is much computationally efficient than equation (6.1).

Based on the equations above, Domain Decomposition can be applied to IDA-MCS, and the example of Domain Decomposition IDA-MCS by 2-by-2 GPU cluster is illustrated in Figure 3. The key idea of Domain Decomposition GPU-cluster IDA-MCS is to divide and distribute the whole IDA-MCS by domain decomposition across multiple nodes then multiple GPU to search the whole space.

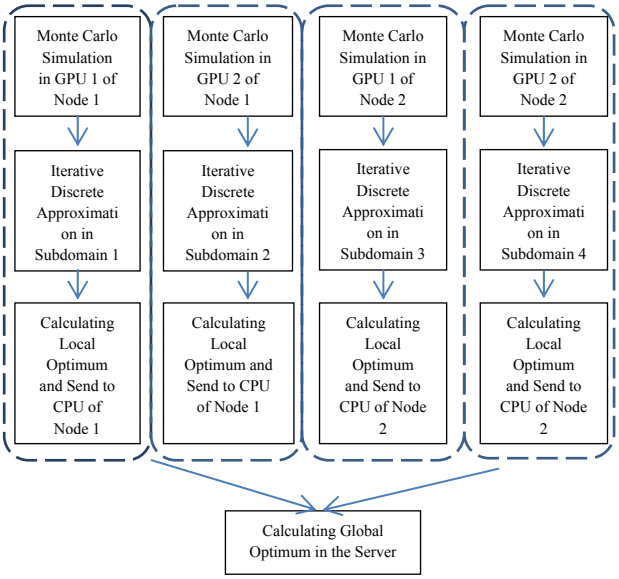


Figure 3. The Framework of Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5) on 2-by-2 GPU Cluster

In Figure 3, searching each subdomain of GPU-cluster IDA-MCS is done in a GPU, which includes three parts: Monte Carlo Simulation, Iterative Discrete Approximation and calculating local optima, and the global optimum is calculated in the server CPU. The details of Domain Decomposition GPU-cluster IDA-MCS are described in Algorithm 5:

Algorithm 5. Domain Decomposition GPU-cluster IDA-MCS

- By the definition of subdomain l, generate a set A_{5l} in GPU m of node n;
- Pass the set A_{5l} through $f(x, y)$ to the set B_{5l};
- In subdomain l and iteration k: discretely approximate the set B_{5l}^{k} to the set C_{5l}^{k};
- In subdomain l and iteration k: pass the set C_{5l}^{k} through equation (3) to the set D_{5l}^{k};
- In subdomain l and iteration k: substitute D_{5l}^{k} back to B_{5l}^{k} for discrete approximation of next iteration k+1;
- In subdomain l, calculate the local optima from the set D_{5l}, and calculate the global optimum from these local optima;

Algorithm 5 begins from the set A_{5l} in each subdomain by GPU m of node n, and passing this set through $f(x, y)$ obtains the set B_{5l}, and this is the step of Monte Carlo Search in subdomain l. Iteratively Discrete Approximating the set B_{5l}^{k} to the set C_{5l}^{k} with equation (3), and passing the set C_{5l}^{k} to the set D_{5l}^{k}. After all iterations finish, the set D_{5l}^{k} is the candidates for calculating the local optima in GPU$_l$, and finally the global optimum is calculated in the server CPU.

F. Local Search GPU-cluster IDA-MCS for 2D Cost Function

Local Search is the other strategy to parallelize IDA-MCS. Local Search means the search space is sampled independently by multiple Iterative Discrete Approximation, and each Iterative Discrete Approximation is also named "particle". Since equation (2) can be rewritten to the formats:

$$\max_{\Omega} f(x,y) = \max_{I} f(x_i, y_i) \quad (7.1)$$

$$= \max\left(\max_{I} f(x_i,y_i)^{(1)}, \max_{I} f(x_i,y_i)^{(2)}, \cdots, \max_{I} f(x_i,y_i)^{(l)}\right) \quad (7.2)$$

each particle calculates a local optimum $\max_{I} f(x_i, y_i)^{(l)}$ in a GPU, and the global optimum is calculated from the set of local optima.

Based on the equations above, we develop Local Search GPU-cluster IDA-MCS of Algorithm 6, and Algorithm 6 performs Local Search of the whole search space in each particle, and the efficiency of this algorithm increases by IDA-MCS in each particle, and an example of Local Search GPU-cluster IDA-MCS (Algorithm 6) on 2-by-2 GPU cluster is shown in Figure 4:

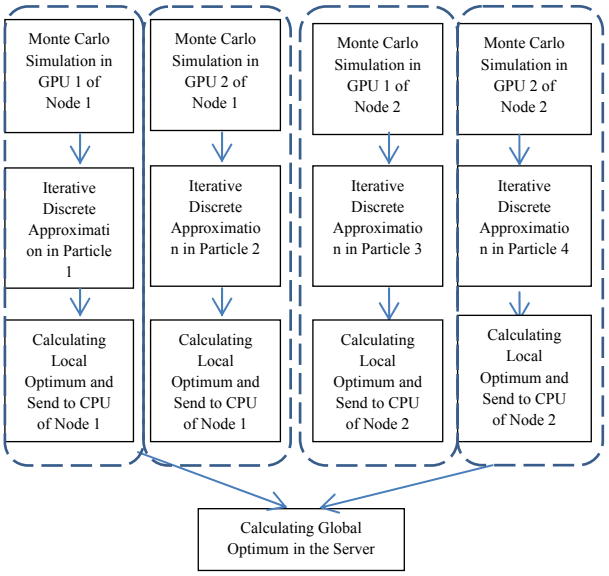


Figure 4. The Framework of Local Search GPU-cluster IDA-MCS (Algorithm 6) on 2-by-2 GPU Cluster

From Figure 4 we can see, in Local Search GPU-cluster IDA-MCS (Algorithm 6), each particle aims at the whole search space, and a local optimum is produced by this particle. Each particle residents in a GPU, and there

is no communication among particles to increase the performance of the algorithm. In each particle of Local Search GPU-cluster IDA-MCS (Algorithm 6), the activities of one iteration include Monte Carlo Simulation, discretely approximating and calculating local optimum, and the global optimum is calculated in the server after all iterations are finished. The details of Local Search GPU-cluster IDA-MCS are in Algorithm 6:

Algorithm 6. Local Search GPU-cluster IDA-MCS

- In particle l, generate a set A_{6l} from the whole search space;
- In particle l, pass the set A_{6l} through $f(x, y)$ to generate the set B_{6l};
- In particle l and iteration k: discretely approximate the set B_{6l}^{k} leading to the set C_{6l}^{k};
- In particle l and iteration k: pass the set C_{6l}^{k} through equation (3) to generate the set D_{6l}^{k};
- In particle l and iteration k: substitute D_{6l}^{k} back to B_{6l}^{k} for discrete approximation of next iteration k+1;
- In particle l, calculate the local optimum from the set D_{6l}, and calculate the global optimum from these local optima;

From Algorithm 6 we can see, Local Search GPU-cluster IDA-MCS (Algorithm 6) begins from generating the set A_{6l}, and pass this set through the cost function $f(x, y)$ to the set B_{6l}, which is the implementation of Monte Carlo Simulation, though there is no local optimum is selected. The set C_{6l} is discretely approximated through equation (3) to the set D_{6l}^{k}, and substitute back to B_{6l}^{k} for next iteration k+1, which is the implementation of Iterative Discrete Approximation. Finally, the global optimum is calculated from the set D_{6l} in the server CPU.

G. Programming Model for M-by-N GPU Cluster System

For a parallel system with M nodes and N GPUs in each node, multiple parallelization strategies can be applied to this system, and we use the CPU-GPU thread based strategy in this paper, and the programming implementation for this system is MPI-CUDA.

In more details, we illustrate CPU-GPU tread based parallelization for M-by-N GPU cluster in Figure 5. As shown in Figure 5, the GPU cluster system consists of a server CPU, multiple node CPUs and multiple GPUs in each node. There is a CPU thread in the server CPU, a CPU thread in node CPU, and a GPU thread in each node GPU.

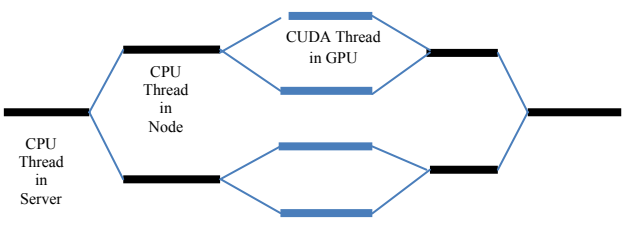


Figure 5. CPU-GPU Thread based Parallelization for M-by-N GPU Cluster System

In more details, the thread of server CPU is responsible for collecting the local optima from each node and calculating the global optimum. The thread of client CPU is responsible for collecting local optima, and managing multiple GPU threads in the node. In the node, each GPU is managed by a GPU thread, and real computation is done in kernels of these GPU threads. In this paper, both Domain Decomposition IDA-MCS (Algorithm 5) and Local Search IDA-MCS (Algorithm 6) are implemented in this parallelization strategy.

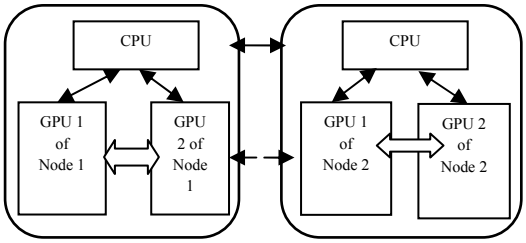


Figure 6. Communication Pattern on 2-by-2 GPU Cluster System

Another problem about GPU cluster is how data is transported? To answer this question, Figure 6 shows the communication pattern on a 2-by-2 GPU cluster. In each node, CPU and GPU are linked by PCI express Intel X79 SandyBridgeE, and GPU and GPU are communicated through PCI expression. CPU in different nodes communicates through 1G switch. GPUs in different nodes cannot be directly communicated except through node CPU.

III. Computational Results

In this section, to test the performance of GPU-cluster IDA-MCS, we code GPU-cluster IDA-MCS by GCC on Fedora 18, CUDA 5.5 and MPICH2, and we test the code of GPU-cluster IDA-MCS on our GPU cluster, which includes four nodes, and each node consists of Intel Core i7-3820 Quad-Core Processor, 8GB memory, four nVidia GTX 660 GPU in each node and Intel X79 SandyBridgeE.

A. The Problem

To test the performance of our methods, we apply the algorithms to an optimization problem with complicated shapes:

$$f(x,y)=abs\left(\frac{x}{a}\times\frac{y}{b}\times\sin(x+y)\right) \qquad (8)$$

$$0\le x\le ub, 0\le y\le ub$$

where ub are the up-bound of optimization variables x and y. The plot of equation (8) with different values of up-bound ub is shown in Figure 7.

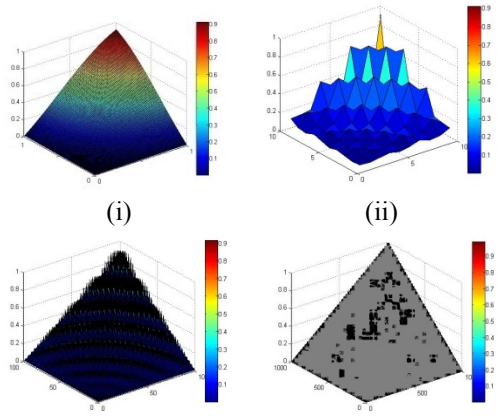


Figure 7. The Plot of the Cost Function of Equation (8) with (i) $0 \le x \le 1$, $0 \le y \le 1$ (ii) $0 \le x \le 10$, $0 \le y \le 10$ (iii) $0 \le x \le 100$, $0 \le y \le 100$ and (iv) $0 \le x \le 1000$, $0 \le y \le 1000$

To find the optima as the "gold standard" for our computational experiments, we optimize equation (8) with different values of up-bound *ub* by Monte Carlo Search (Algorithm 1) with the bandwidth $N = 10^8$ in a CPU server with 64G memory. We estimate the number of peaks in each plot, and the results are listed in Table 1.

TABLE 1.
THE RESULTS OF MONTE CARLO SIMULATION (ALGORITHM 1) FOR THE COST FUNCTION EQUATION (8)

Search Space		Maximum Value	Number of Peaks
$0 \le x \le 1$	$0 \le y \le 1$	0.909294203757878	1
$0 \le x \le 10$	$0 \le y \le 10$	0.912912611660599	22
$0 \le x \le 100$	$0 \le y \le 100$	0.963878552508647	3058
$0 \le x \le 1000$	$0 \le y \le 1000$	0.999624257749903	317038

As listed in Table 1, increasing the up-bounds *ub* leads to much larger numbers of peaks, the optimum of the cost function equation (8) approaches the value of one, and the computational time becomes longer and longer. Actually, to approximate the maximum value of equation (8) with *ub* = 1000, Monte Carlo Search (Algorithm 1) with the bandwidth $N = 10^8$ spends hours in the CPU server with 64G memory. The maximum values obtained by this method are treated as the "gold standard" of our evaluating algorithms of ICA-MCS.

B. The Performance of Domain Decomposition GPU-cluster IDA-MCS

To test the performance of Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5), we apply this method with the bandwidth $N = 10^4$ to optimize the cost function of equation (8) with *ub* =1000 by the discretization of 4-by-4 (4 domain then 4 subdomain). Domain Decomposition IDA-MCS (Algorithm 5) independently runs for three times with ten iterations, the global optimum is plotted against the "gold standard" from Table 1, and the results are plotted in Figure 8.

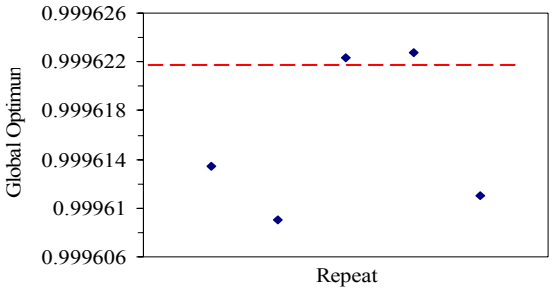


Figure 8. The Global Optimum of Domain Decomposition IDA-MCS (Algorithm 5) against Monte Carlo Search (Algorithm 1)

From Figure 8 we can see, the global optimum from Domain Decomposition IDA-MCS (Algorithm 5) closely approaches the "gold standard" from Monte Carlo Search (Algorithm 1) in Table 1, which is the situation of the last iteration of Domain Decomposition IDA-MCS (Algorithm 5). To illustrate the situation of each iteration, we run three repeats of Domain Decomposition IDA-MCS (Algorithm 5), and plot the values against the "gold standard" from Monte Carlo Search (Algorithm 1) in Figure 9.

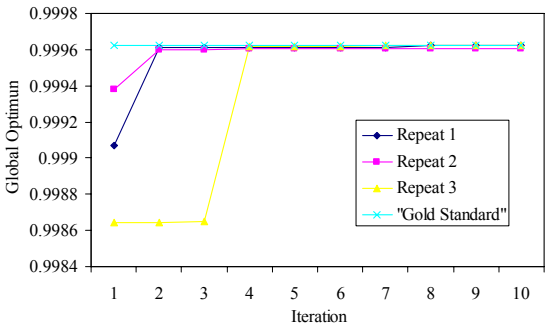


Figure 9. The Performance of Domain Decomposition IDA-MCS (Algorithm 5) in each Iteration against Monte Carlo Search (Algorithm 1)

In Figure 9, while the "gold standard" is plotted with the line of cross, the first repeat of Domain Decomposition IDA-MCS (Algorithm 5) is plotted with the line of diamond, the second repeat of IDA-MCS (Algorithm 5) is plotted with line of square, and the third repeat of IDA-MCS (Algorithm 5) is plotted with the line of triangle. From iteration 4, the diamond line, the square line and the triangle line closely approach the cross line, that is to say, GPU-cluster IDA-MCS (Algorithm 5) with bandwidth $N = 10^4$ is as competitive as Monte Carlo Search (Algorithm 1) with bandwidth $N = 10^8$.

C. The Performance of Local Search GPU-cluster IDA-MCS

To study the performance of Local Search GPU-cluster IDA-MCS (Algorithm 6) against Monte Carlo Search (Algorithm 1), same to Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5), we run three repeats of Local Search IDA-MCS (Algorithm 6), and calculated global optima are plotted in Figure 10.

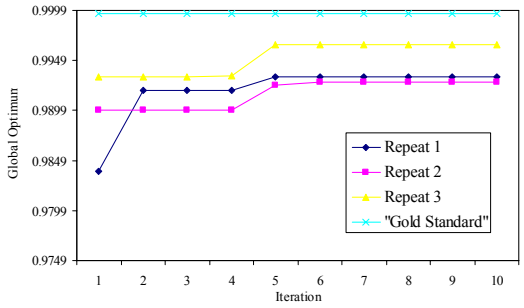


Figure 10. The Performance of Local Search GPU-cluster IDA-MCS (Algorithm 6) in each Iteration against Monte Carlo Search (Algorithm 1)

From Figure 10 we can see, the global optimum from each repeat of Local Search IDA-MCS (Algorithm 6) approaches the "gold standard" along with the iterations. However, there is still obvious difference between the global optimum from Local Search IDA-MCS (Algorithm 6) and the "gold standard" from Monte Carlo Search (Algorithm 1).

D. Presentation of Accuracy

Computational results above present the performance of IDA-MCS algorithms by plots. To quantitative measure the performance of these IDA-MCS algorithms, based on the "gold standard" from Monte Carlo Search (Algorithm 1), we develop accumulative accuracy in Table 2.

TABLE 2.
ACCUMULATIVE ACCURACY OF MULTIPLE REPEAT OF GLOBAL OPTIMIZATION FROM ICA-MCS ALGORITHMS

Accuracy	-8	-7	-6	-5	-4	-3	-2	-1
1	0	0	0	0	1	1	1	1
2	0	0	0	0	1	1	1	1
3	0	0	0	1	1	1	1	1
4	0	0	0	1	1	1	1	1
5	0	0	0	0	1	1	1	1
Total	0	0	0	3	5	5	5	5

How accumulative accuracy work? In Figure 8, we plot the results from five repeats of Domain Decomposition IDA-MCS (Algorithm 5) with 4-by-4 discretization against the "gold standard" from Monte Carlo Search (Algorithm 1). To quantitatively represent these results against the "gold standard" from Monte Carlo Simulation (Algorithm 1), we take following steps:

Step 1: Write down the results and the "gold standard": 0.99961340 (Repeat 1), 0.99960899 (Repeat 2), 0.99962229 (Repeat 3), 0.99962276 (Repeat 4), 0.99961102 (Repeat 5) and 0.99962426 (the "gold standard");

Step 2: From the last digital of a result from Domain Decomposition IDA-MCS (Algorithm 5), compare against the corresponding digital of the "gold standard". If the digital of a result is bigger than the corresponding digital of the "gold standard", then write 1 in Table 2, else write 0;

Step 3: After all results is compared against the "gold standard", summarize the "1" in each digital as the accumulative accuracy of IDA-MCS algorithms.

Table 2 shows that, comparing with the "gold standard" from Monte Carlo Search (Algorithm 1), five repeats of Domain Decomposition IDA-MCS (Algorithm 5) with 4-by-4 discretization shown in Figure 8 obtain three times of 10^{-5}, five times of 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1}.

E. Domain Decomposition v.s. Local Search for GPU-cluster IDA-MCS

Since we apply two different parallelization strategies, Domain Decomposition and Local Search, to IDA-MCS, a natural question may be: which one is better? To answer this question, we run Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5) for ten repeats with a 16-by-16 decomposition, and Local Search GPU-cluster IDA-MCS (Algorithm 6) for ten times, measure their performance by the method described in the section of 3.4, and the results are plotted in Figure 11.

In Figure 11, both Domain Decomposition IDA-MCS (Algorithm 5) and Local Search IDA-MCS (Algorithm 6) are applied to optimize the cost function of equation (8) with ub = 1000, which consists of 317038 peaks with such setting.

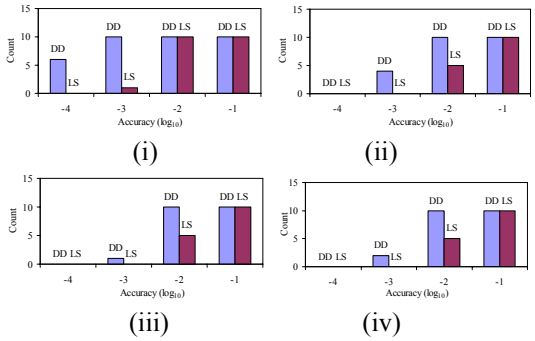


Figure 11. The Accumulative Accuracy of Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5) with (i) N =10000 and ub = 1000 (ii) N = 1000 and ub = 1000 (iii) N = 100 and ub = 1000 (iv) N = 10 and ub = 1000

In Figure 11, both methods are measured by four levels of accumulative accuracy: 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1} on x-axis. In each level of accumulative accuracy, Domain Decomposition IDA-MCS (Algorithm 5) is plotted on the left bar and Local Search IDA-MCS (Algorithm 6) is plotted on the right bar. To bandwidth of each method is: N = 10000 in Figure 11 (i), N = 1000 in Figure 11 (ii), N = 100 in Figure 11 (iii) and N = 10 in Figure 11 (iv). From Figure 11 we can see, the performance of Domain Decomposition ICA-MCS (Algorithm 5) is significantly better than Local Search ICA-MCS (Algorithm 6).

To further compare the performance between Domain Decomposition IDA-MCS (Algorithm 5) and Local Search IDA-MCS (Algorithm 6), we apply these two methods to optimize the cost function of equation (8) with ub = 100, which consists of 3058 peaks with such setting. Since the number of peaks is much smaller, both methods are measured by four levels of accumulative accuracy from 10^{-8} to 10^{-1} on x-axis, and the results are plotted in Figure 12.

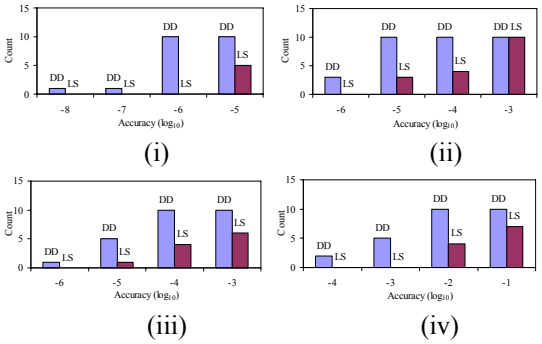


Figure 12. The Accumulative Accuracy of Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5) with (i) N = 10000 and ub = 100 (ii) N =1000 and ub = 100 (iii) N = 100 and ub = 100 (iv) N = 10 and ub = 100

From Figure 12 we can see, the performance of Domain Decomposition ICA-MCS (Algorithm 5) is again significantly better than Local Search ICA-MCS (Algorithm 6) with the up-bound of 100. The accumulative accuracy of both methods decreases from Figure 12 (i) to Figure 12 (iv), because the bandwidth of both methods decreases.

F. The Scalability of Domain Decomposition GPU-cluster IDA-MCS

Since Domain Decomposition IDA-MCS (Algorithm 5) is built on GPU cluster, the accumulative accuracy of Domain Decomposition IDA-MCS (Algorithm 5) should be scalable to the number of GPUs. To test the scalability of Domain Decomposition IDA-MCS (Algorithm 5) with respect to the accumulative accuracy, we apply Domain Decomposition IDA-MCS (Algorithm 5) to the cost function equation (8) with the number of GPUs 16 (4 nodes, Algorithm 5), 12 (3 nodes, Algorithm 5), 8 (2 nodes, Algorithm 5), 4 (1 node, Algorithm 3) and 1 (Algorithm 2), and the results are plotted in Figure 13.

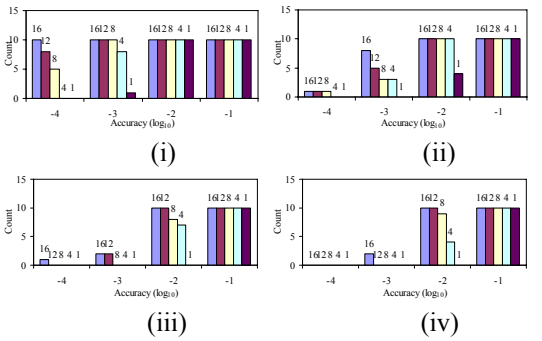


Figure 13. The Scalability of Domain Decomposition IDA-MCS (Algorithm 5) with (i) N = 10000 and ub = 1000 (ii) N = 1000 and ub = 1000 (iii) N = 100 and ub = 1000 (iv) N = 10 and ub = 1000

From Figure 13 we can see, larger number of GPUs leads to higher accumulative accuracy of IDA-MCS than smaller number of GPUs. In Figure 13 (i), Domain Decomposition IDA-MCS (Algorithm 5) of 16 GPU delivers the best performance as high as 10^{-4}, while that of 4 GPU (Algorithm 3) performs much poor in the accumulative accuracy of 10^{-4}. Similar situation happens in Figure 13 (ii), Figure 13 (iii) and Figure 13 (iv).

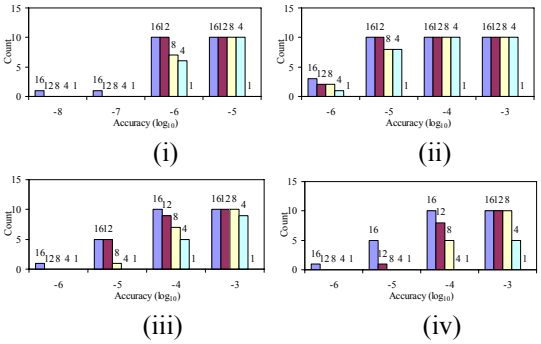


Figure 14. The Scalability of Domain Decomposition GPU-cluster IDA-MCS (Algorithm 5) with (i) N = 10000 and ub = 100 (ii) N = 1000 and ub = 100 (iii) N = 100 and ub = 100 (iv) N =10 and ub = 100

We also test the scalability of Domain Decomposition IDA-MCS (Algorithm 5) for the cost function equation (8) with up-bound ub = 100, and the results are plotted in Figure 14. For Figure 14 we can see, for the cost function equation (8) with ub = 100, increasing the number of GPUs significantly increase the accumulative accuracy of IDA-MCS (Algorithm 5).

G. The Scalability of Local Search GPU-cluster IDA-MCS

Since Local Search IDA-MCS (Algorithm 6) is built on GPU cluster, the accumulative accuracy of Local Search IDA-MCS (Algorithm 6) should be scalable to the size of GPU cluster. To test the scalability of Local Search IDA-MCS (Algorithm 6) with respect to the accumulative accuracy, we apply Local Search IDA-MCS (Algorithm 6) to the cost function equation (8) with the number of GPUs 16 (4 nodes, Algorithm 6), 12 (3 nodes, Algorithm 6), 8 (2 nodes, Algorithm 6), 4 (1 node, Algorithm 4) and 1 (Algorithm 2), and the results are plotted in Figure 15.

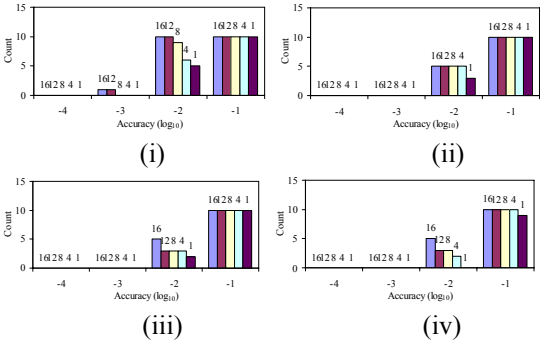


Figure 15. The Scalability of Local Search GPU-cluster IDA-MCS (Algorithm 6) with (i) N = 10000 and ub = 1000 (ii) N = 1000 and ub = 1000 (iii) N = 100 and ub = 1000 (iv) N = 10 and ub = 1000

From Figure 15 we can see, larger number of GPUs leads to higher accumulative accuracy of Local Search IDA-MCS (Algorithm 6) than smaller number of GPUs. In Figure 15 (i), Local Search IDA-MCS (Algorithm 6) of 16 GPU delivers the best performance as high as 10^{-4}, while that of 4 GPU (Algorithm 4) performs much poor in the accuracy of 10^{-4}. Similar situation happens in Figure 15 (ii), Figure 15 (iii) and Figure 15 (iv).

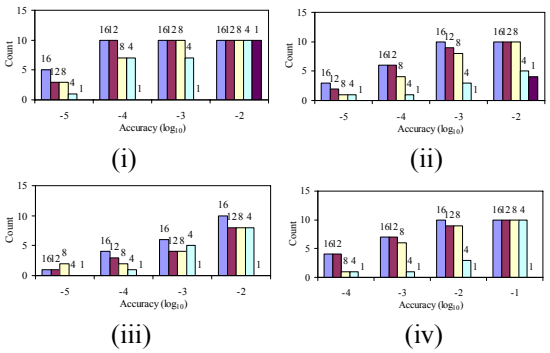


Figure 16. The Scalability of Local Search GPU-cluster IDA-MCS with (i) N = 10000 and ub = 100 (ii) N = 1000 and ub = 100 (iii) N = 100 and ub = 100 (iv) N = 10 and ub = 100

We also test the scalability of Local Search IDA-MCS (Algorithm 6) for the cost function equation (8) with up-bound *ub* = 100, and the results are plotted in Figure 16. For Figure 16 we can see, for the cost function equation (8) with *ub* = 100, increasing number of GPUs significantly increases the accumulative accuracy of Local Search IDA-MCS (Algorithm 6).

H. Effects of Grid Size on Domain Decomposition IDA-MCS

In previous computational experiments of Domain Decomposition IDA-MCS (Algorithm 5), we use the grid size of 4-by-4 discretization (4 domain then 4 subdomain). How the selection of grid size affects the performance of Domain Decomposition IDA-MCS (Algorithm 5)? To answer this question, we test Domain Decomposition IDA-MCS (Algorithm 5) by the grid size of 4-by-4, 8-by-8, 12-by-12 and 16-by-16 discretization, and the results are plotted in Figure 17.

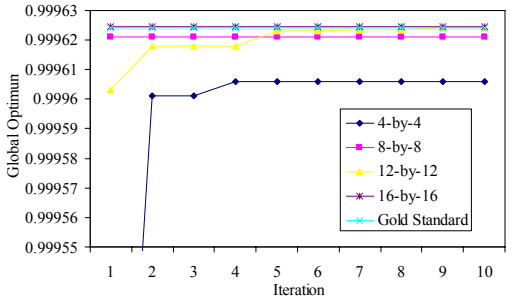


Figure 17. Effects of The Accumulative Accuracy from Domain Decomposition IDA-MCS (Algorithm 5) on the Selection of Grid Sizes

From Figure 17 we can see, increasing the grid size significantly increases the performance of Domain Decomposition IDA-MCS (Algorithm 5). While Domain Decomposition with the grid size 4-by-4 discretization approaches the "gold standard" from Monte Carlo Simulation (Algorithm 1) in Table 1, the Domain Decomposition IDA-MCS (Algorithm 5) with the grid size 16-by-16 discretization almost exactly matches the "gold standard" from Monte Carlo Simulation (Algorithm 1).

IV. Discussion

In this paper, we develop two-dimensional IDA-MCS, we parallelize this algorithm by Domain Decomposition and Local Search, and we implement this algorithm on a GPU cluster. Computational results show a higher accuracy and efficiency of IDA-MCS than the conventional Monte Carlo Search.

Since the algorithm IDA-MCS is built on Monte Carlo Simulation, the efficiency of IDA-MCS may be improved by better samplers such as Markov Chain Monte Carlo and Sequential Monte Carlo with multiple Markov chains, and better approximation of the cost function may be provided.

We design a weighting mechanism equation (3) to increase the performance of IDA-MCS, and we use equation (5) in this paper. However, the power function equation (5) is not the only design for equation (3), and other approaches such as the exponentiation function can also be choosed for this purpose.

In this paper, we construct Iterative Discrete Approximation by measuring the value of the cost function. However, the value of the cost function can be replaced by quantities which represent the changes of the cost function such as integration of a finite area in the search space, and then constructing bins by these quantities.

To parallelize IDA-MCS, we employ two different parallel schemes: Domain Decomposition and Local Search, and computational results show that Domain Decomposition is much better than Local Search. However, new parallel schemes can be designed to parallel algorithms of IDA-MCS. For example, instead of current design of limiting IDA-MCS in each GPU, we are working on expanding IDA-MCS across the GPU cluster.

In this paper, two parallel schemes Domain Decomposition and Local Search are coded by MPICH2, and there is no direction communication between GPUs in different nodes, which decreases the performance of IDA-MCS. By CUDA-Aware MPI such as MVAPICH2 [40-44], Remote Direct GPU Memory Access is possible to improve the performance of data communication in IDA-MCS.

We developed one-dimensional IDA-MCS in our previous research, and we develop two-dimensional IDA-MCS in this paper. Can IDA-MCS be applied to higher dimensional problem such as three-dimension? To realize this goal, the theory and the implementation methods of IDA-MCS should be developed.

In our previous research and this paper, IDA-MCS is applied to a traditional problem of applied mathematics optimization. In future research, we may expand IDA-MCS to more problems such as integration and mesh generation. For example, since IDA-MCS can concentrate samples on the peak area, triangulation of these points produces mesh with high density in peak area which needs more computation, and low density in flat area which needs less computation.

References

[1] You, Z. and Ying, T. *GPU-based parallel particle swarm optimization*. City, 2009.

[2] Mussi, L., Nashed, Y. S. G. and Cagnoni, S. GPU-based asynchronous particle swarm optimization. In *Proceedings of the Proceedings of the 13th annual conference on Genetic and evolutionary computation* (Dublin, Ireland, 2011). ACM, [insert City of Publication],[insert 2011 of Publication].
[3] Mussi, L., Daolio, F. and Cagnoni, S. Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture. *Information Sciences*, 181, 20 2011), 4642-4657.
[4] Rabinovich, M., Kainga, P., Johnson, D., Shafer, B., Lee, J. J. and Eberhart, R. *Particle Swarm Optimization on a GPU*. City, 2012.
[5] Hung, Y. and Wang, W. Accelerating parallel particle swarm optimization via GPU. *Optimization Methods and Software*, 27, 1 (2012/02/01 2010), 33-51.
[6] Cagnoni, S., Bacchini, A. and Mussi, L. *OpenCL Implementation of Particle Swarm Optimization: A Comparison between Multi-core CPU and GPU Performances*. Springer Berlin Heidelberg, City, 2012.
[7] Calazan, R. M., Nedjah, N. and de Macedo Mourelle, L. *Parallel GPU-based implementation of high dimension Particle Swarm Optimizations*. City, 2013.
[8] Calazan, R., Nedjah, N. and Macedo Mourelle, L. *Three Alternatives for Parallel GPU-Based Implementations of High Performance Particle Swarm Optimization*. Springer Berlin Heidelberg, City, 2013.
[9] Souza, D., Teixeira, O., Monteiro, D. and Oliveira, R. *A New Cooperative Evolutionary Multi-Swarm Optimizer Algorithm Based on CUDA Architecture Applied to Engineering Optimization*. Springer Berlin Heidelberg, City, 2013.
[10] Mehmood, S., Cagnoni, S., Mordonini, M. and Khan, S. An embedded architecture for real-time object detection in digital images based on niching particle swarm optimization. *J Real-Time Image Proc*(2012/06/06 2012), 1-15.
[11] Ugolotti, R., Nashed, Y. S. G., Mesejo, P., Ivekovič, Š., Mussi, L. and Cagnoni, S. Particle Swarm Optimization and Differential Evolution for model-based object detection. *Applied Soft Computing*, 13, 6 2013), 3092-3105.
[12] Calazan, R., Nedjah, N. and Macedo Mourelle, L. *Swarm Grid: A Proposal for High Performance of Parallel Particle Swarm Optimization Using GPGPU*. Springer Berlin Heidelberg, City, 2012.
[13] Hsieh, T.-J., Yang, Y.-S., Wang, J.-H. and Shen, W.-J. Feature extraction using bionic particle swarm tracing for transfer function design in direct volume rendering. *Vis Comput*(2013/02/12 2013), 1-12.
[14] Mussi, L., Cagnoni, S. and Daolio, F. *GPU-Based Road Sign Detection Using Particle Swarm Optimization*. City, 2009.
[15] Sharma, B., Thulasiram, R. and Thulasiraman, P. Normalized particle swarm optimization for complex chooser option pricing on graphics processing unit. *J Supercomput*(2013/02/28 2013), 1-23.
[16] Sharma, B., Thulasiram, R. K. and Thulasiraman, P. *Portfolio Management Using Particle Swarm Optimization on GPU*. City, 2012.
[17] Bo, Z., Zheng-hui, X., Wei-ming, L., Wu, R. and Xin-qing, S. *Particle Swarm Optimization of frequency selective surface*. City, 2012.
[18] Nobile, M. S., Besozzi, D., Cazzaniga, P., Mauri, G. and Pescini, D. Estimating reaction constants in stochastic biological systems with a multi-swarm PSO running on GPUs. In *Proceedings of the Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion* (Philadelphia, Pennsylvania, USA, 2012). ACM, [insert City of Publication],[insert 2012 of Publication].
[19] Nakagawa, T., Iwahori, Y. and Bhuyan, M. K. *Defect Classification of Electronic Board Using Multiple Classifiers and Grid Search of SVM Parameters*. Springer International Publishing, City, 2013.
[20] Tian, W., Xiufen, Y., Lei, W. and Heyi, L. *Grid Search Optimized SVM Method for Dish-like Underwater Robot Attitude Prediction*. City, 2012.
[21] Zhou, L., Lai, K. and Yu, L. Credit scoring using support vector machines with direct search for parameters selection. *Soft Comput*, 13, 2 (2009/01/01 2009), 149-155.
[22] Jinghua, L., Congying, Z. and Zhenning, L. *Battlefield Target Identification Based on Improved Grid-Search SVM Classifier*. City, 2009.
[23] Hongtao, Z., Shuping, Y. and Yuxia, H. *SVM Classifier of Stored-Grain Insects Based on Grid Search*. Springer Berlin Heidelberg, City, 2011.
[24] Bao, Y. and Liu, Z. *A Fast Grid Search Method in Support Vector Regression Forecasting Time Series*. Springer Berlin Heidelberg, City, 2006.
[25] Jiménez, Á., Lázaro, J. and Dorronsoro, J. *Finding Optimal Model Parameters by Discrete Grid Search*. Springer Berlin Heidelberg, City, 2007.
[26] Barbero Jiménez, Á., López Lázaro, J. and Dorronsoro, J. R. Finding optimal model parameters by deterministic and annealed focused grid search. *Neurocomputing*, 72, 13–15 2009), 2824-2832.
[27] Chen, W., Ma, C. and Ma, L. Mining the customer credit using hybrid support vector machine technique. *Expert Systems with Applications*, 36, 4 2009), 7611-7616.
[28] Min, J. H. and Lee, Y.-C. Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. *Expert Systems with Applications*, 28, 4 2005), 603-614.
[29] Dipama, J., Teyssedou, A., Aubé, F. and Lizon-A-Lugrin, L. A grid based multi-objective evolutionary algorithm for the optimization of power plants. *Applied Thermal Engineering*, 30, 8–9 2010), 807-816.
[30] Omata, K., Hashimoto, M., Sutarto and Yamada, M. Artificial Neural Network and Grid Search Aided Optimization of Temperature Profile of Temperature Gradient Reactor for Dimethyl Ether Synthesis from Syngas. *Industrial & Engineering Chemistry Research*, 48, 2 (2009/01/21 2008), 844-849.
[31] Piehowski, P. D., Petyuk, V. A., Sandoval, J. D., Burnum, K. E., Kiebel, G. R., Monroe, M. E., Anderson, G. A., Camp, D. G. and Smith, R. D. STEPS: A grid search methodology for optimized peptide identification filtering of MS/MS database search results. *PROTEOMICS*, 13, 5 2013), 766-770.
[32] Azar, A. and El-Said, S. Performance analysis of support vector machines classifiers in breast cancer mammography recognition. *Neural Comput & Applic*(2013/01/24 2013), 1-15.
[33] Chen, H.-L., Yang, B., Wang, G., Wang, S.-J., Liu, J. and Liu, D.-Y. Support Vector Machine Based Diagnostic System for Breast Cancer Using Swarm Intelligence. *J Med Syst*, 36, 4 (2012/08/01 2012), 2505-2519.
[34] Donahue, M. M., Buzzard, G. T. and Rundell, A. E. *Robust parameter identification with adaptive sparse grid-based optimization for nonlinear systems biology models*. City, 2009.
[35] Wang, J., Du, H., Yao, X. and Hu, Z. Using classification structure pharmacokinetic relationship (SCPR) method to predict drug bioavailability based on grid-search support

vector machine. *Analytica Chimica Acta*, 601, 2 2007), 156-163.
[36] Wei-Chang, Y., Yi-Cheng, L., Chung, Y. Y. and Mingchang, C. A Particle Swarm Optimization Approach Based on Monte Carlo Simulation for Solving the Complex Network Reliability Problem. *Reliability, IEEE Transactions on*, 59, 1 2010), 212-221.
[37] Widder, J., Hollander, M., Ubbels, J. F., Bolt, R. A. and Langendijk, J. A. Optimizing dose prescription in stereotactic body radiotherapy for lung tumours using Monte Carlo dose calculation. *Radiotherapy and Oncology*, 94, 1 2010), 42-46.
[38] Yu, C., Li, R., He, Q., Yu, L. and Tan, J. *Fault Diagnosis of Nodes in WSN Based on Particle Swarm Optimization*. Springer Berlin Heidelberg, City, 2013.
[39] Mukhopadhyay, A. and Mandal, M. *A Hybrid Multiobjective Particle Swarm Optimization Approach for Non-redundant Gene Marker Selection*. Springer India, City, 2013.
[40] Bureddy, D., Wang, H., Venkatesh, A., Potluri, S. and Panda, D. K. *OMB-GPU: A Micro-Benchmark Suite for Evaluating MPI Libraries on GPU Clusters*. Springer Berlin Heidelberg, City, 2012.
[41] Potluri, S., Wang, H., Bureddy, D., Singh, A. K., Rosales, C. and Panda, D. K. *Optimizing MPI Communication on Multi-GPU Systems Using CUDA Inter-Process Communication*. City, 2012.
[42] Wang, H., Potluri, S., Luo, M., Singh, A., Sur, S. and Panda, D. MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters. *Comput Sci Res Dev*, 26, 3-4 (2011/06/01 2011), 257-266.
[43] Hao, W., Potluri, S., Miao, L., Singh, A. K., Xiangyong, O., Sur, S. and Panda, D. K. *Optimized Non-contiguous MPI Datatype Communication for GPU Clusters: Design, Implementation and Evaluation with MVAPICH2*. City, 2011.
[44] Singh, A. K., Potluri, S., Hao, W., Kandalla, K., Sur, S. and Panda, D. K. *MPI Alltoall Personalized Exchange on GPGPU Clusters: Design Alternatives and Benefit*. City, 2011.