Particle Swarm Stepwise Algorithm (PaSS) on Multicore Hybrid CPU-GPU Clusters

Mu Yang
Institute of Applied Mathematical Sciences, National Taiwan University, Taipei, Taiwan
Email: muyang@ntu.edu.tw

Ray-Bing Chen
Department of Statistics, National Cheng-Kung University, Tainan, Taiwan
Email: rbchen@stat.ncku.edu.tw

I-Hsin Chung
IBM Thomas J. Watson Research Center, New York, United States
Email: ihchung@us.ibm.com

Weichung Wang
Institute of Applied Mathematical Sciences, National Taiwan University, Taipei, Taiwan
Email: wwang@ntu.edu.tw

***Abstract*—Variable (feature) selection is a key component in artificial intelligence. One way to perform variable selection is to solve the information-criterion-based optimization problems. These optimization problems arise from data mining, genomes analysis, machine learning, numerical simulations, and others. Particle Swarm Stepwise Algorithm (PaSS) is a stochastic search algorithm proposed to solve the information-criterion-based variable selection optimization problems. It has been shown recently that the PaSS outperforms several existed methods. However, to solve the target optimization problems remains a challenge due to the large search spaces. We tackle this issue by proposing a parallel version of the PaSS on clusters equipped with CPU and GPU to shorten the computational time without compromise in solution accuracy. We have successfully achieved near-linear scalability on CPU with single to 64 threads and gained further 7X faster timing performance by using GPU.**

***Keywords*-information criterion; variable selection ensemble; optimization; parallel computing; hybrid CPU/GPU cluster, OpenMP; MPI; GPU;**

I. Introduction

We consider information-criterion-based variable selection problems in this article. Variable selection is a kernel of many applications with big datasets. These applications include text mining of (on-line) documents, genomes expression analysis, machine learning, and numerical simulations, just to name a few. By suitably selecting the effective variables among many possibilities, we are able to improve the effectiveness of a predictor (or a prediction model), simplify the predictor, enhance the prediction accuracy, reduce training and prediction time, and explore latent structure of a dataset. Information criteria are used to evaluate how good a predictor is based on a certain set of selected variables. By introducing an information-criterion, a variable selection problem can be formulated as an optimization problem. Solving such optimization problem is not a trivial, as the number of variables and the size of the corresponding datasets grow rapidly. In other words, we need to solve an optimization problem with large model space. We start our discussion from the definitions of the problems.

Given a dataset of a response variable, y and a set of the candidate predictors, $\mathbf{x} = (x_1, \ldots, x_p)^\top$, the goal of the variable (feature) selection is to identify the subset of the important variables, $x_1^*, \ldots, x_q^*$ to describe the functional relationship among the response and variables, $y \approx f(x_1^*, \ldots, x_q^*)$. Following the sparse model assumptions [1][2], variable selection essentially plays an important role in data analyzes especially when we have a hug number of the candidate variables.

In this article, we consider the variable selection problem defined by the linear regression model

$$Y = \mathbf{X}\beta + \epsilon. \tag{1}$$

Here, Y is an $n \times 1$ response vector, $\mathbf{X} = [X_1, \ldots, X_p]$ is an $n \times p$ model matrix, X_i is the i-th predictor variable (or regressor), $\beta = (\beta_1, \ldots, \beta_p)^\top$ is a $p \times 1$ vector of the unknown coefficients, and $\epsilon = (\epsilon_1, \ldots, \epsilon_n)^\top$ is an $n \times 1$ noise vector that follows a multivariate normal distribution with zero mean vector and covariance matrix $\sigma^2 I_n$.

In Statistics, various information criteria are widely used to measure how good a *model* (i.e., a subset of the variables) is. Akaike [3] proposed the Akaike information criterion (AIC) to select the variables by minimizing the Kullback-Leibler (KL) divergence of the selected model and the true model and AIC can be represented as $\mathrm{AIC}(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + 2\#(\mathcal{T})$, where $\mathcal{T} \subseteq \{1, \ldots, p\}$ is an non-empty set, $\#(\mathcal{T})$ is the number of the components in J, $\hat{\sigma}^2_{\mathcal{T}} = n^{-1}||Y - \hat{Y}_{\mathcal{T}}||_2^2$ is the 2-norm prediction error. Another popular information criterion is the Bayesian information criterion (BIC) proposed by Schwarz [4]. The AIC and BIC fail for large-p-small-n selection problems; therefore, Chen and Chen [5] proposed an extended BIC (EBIC) when $p > n$, and Ing and Lai [6] also introduced other high-dimensional information criteria based on a sparse model assumption. Overall the information criterion is the combination of the fitness of the model with a penalty term for the model complexity, and given an information criterion, the smaller criterion variable is, the better model is. Thus information-criterion-based variable selection problem is to find the model (a subset of variables) with the smallest

978-1-5090-4314-9/16 $25.00 © 2016 IEEE
DOI 10.1109/CIT.2016.101

pre-specified information criterion value among all possible models.

Here the information-criterion-based variable selection problem can then be formulated as the optimization problem

$$\mathcal{T}^* = \arg\min_{\mathcal{T}} \Phi(\mathcal{T}) \tag{2}$$

corresponding to an information criterion, where, $\Phi(\cdot)$ is a pre-specified information criterion, and $\mathcal{T}$ is a model. To solve this optimization problem, we use a stochastic algorithm called Particle Swarm Stepwise Algorithm (PaSS) that uses multiple particles as search agents to find out the optimal solution. Each particle updates the search result via a stepwise procedure that adds new variables to forward selection and deletes variables to backward selection.

Our main contributions for tackling these challenging information-criterion-based variable selection problems are two-fold. First, we study how to efficiently implement this PaSS on multicore clusters, discuss how to use multithread and multicore parallelization to speed up the computation, and introduce the GPU acceleration on hybrid CPU-GPU architecture. Second, we conduct intensive numerical experiments to demonstrate the promising computational properties of PaSS regarding prediction accuracy, computational timing, parallelism scalability, and parameter tuning.

II. Related Work

The problem shown in (2) is a discrete optimization problem, and can be treated as a L_0 regularization problem. Since the solution domain contains 2^p candidate models generated from the p variables, it is impractical to evaluate each of the candidate models especially for large p, and thus adaptive selection procedures can be combined with L_1 regularization or a greedy algorithm. For example, Lasso [1] and SCAD [7] are each combined with the all-subset selection for the EBIC in Chen and Chen [5]. Ing and Lai [6] proposed another procedure wherein the orthogonal greedy algorithm is integrated with high-dimensional information criteria for high-dimensional sparse linear models. Consider the L_0 regularization problem. Zhang [8] proposed the adaptive forward-backward (FoBa) greedy algorithm to learn the linear sparse model and this FoBa can also be used to solve (2). Chen et al. [9] proposed an efficient algorithm, particle swarm stepwise (PaSS) algorithm to solve the variable selection problem (2) by integrating FoBa algorithm and the particle swarm optimization (PSO) [10]. In Chen et al. [9], they have demonstrated that PaSS can identify the best model for the pre-specified information criterion by efficiently solving the variable selection problem (2). They have also shown the advantage of PaSS by comparing PaSS with several existed methods.

III. Particle Swarm Stepwise Algorithm (PaSS)

The difficulty in solving the information-criterion-based variable selection problem (2) is mainly due to the large size of the model space. Particle Swarm Stepwise algorithm (PaSS), proposed by [9], is to address this issue based on the idea of the discrete version of the particle swarm optimization (PSO). The details of PaSS is introduced in the following.

1: Generate initial models, $\mathcal{T}_j^0$, $j = 1, \ldots, K$.
2: Compute function value $\Phi(\mathcal{T}_j^0)$ of each model, $\mathcal{T}_j^0$.
3: Initialize the current best model $\mathcal{T}_{\rm best} = \arg\min_{\mathcal{T}_j^0} \Phi(\mathcal{T}_j^0)$.
4: Let s_j^i be the indicator that takes the value -1 or 1. $s_j^i = 1$ means that the model $\mathcal{T}_j$ is updated by the forward step at the i-th iteration, and $s_j^i = -1$ means that $\mathcal{T}_j$ is updated by the backward step.
5: Define $\mathcal{T} = \{1, \ldots, p\}$.
6: **for** each iteration i **do**
7: **for** each particle j **do**
8: // Decide to implement forward or backward step
9: **if** $i = 1$ or $|\mathcal{T}_j^i| = 1$ **then**
10: $s_j^i = 1$
11: **end if**
12: **if** $\Phi(\mathcal{T}_j^{i-1}) < \Phi(\mathcal{T}_j^{i-2})$ **then**
13: $s_j^i = s_j^{i-1}$
14: **else**
15: $s_j^i = -s_j^{i-1}$
16: **end if**
17: // Update each model
18: **if** $s_j^i = 1$ **then**
19: Implement forward step, $\mathcal{T}_j^i = \mathcal{T}_j^{i-1} \cup \{\mathrm{i}^{\rm f}\}$, where
$$\mathrm{i}^{\rm f} = \begin{cases} \mathrm{i}^{\rm f}_{\rm best} \in (\mathcal{T} \setminus \mathcal{T}_j^{i-1}) \cap \mathcal{T}_{\rm best} & \text{with probability } p^{\rm f}_{\rm b} \\ \mathrm{i}^{\rm f}_{\rm impr} \in \mathcal{T} \setminus \mathcal{T}_j^{i-1} & \text{with probability } p^{\rm f}_{\rm i} \\ \mathrm{i}^{\rm f}_{\rm rand} \in \mathcal{T} \setminus \mathcal{T}_j^{i-1} & \text{with probability } p^{\rm f}_{\rm r} \end{cases}$$
20: **end if**
21: **if** $s_j^i = -1$ **then**
22: Implement backward step, $\mathcal{T}_j^i = \mathcal{T}_j^{i-1} \setminus \{\mathrm{i}^{\rm b}\}$, where
$$\mathrm{i}^{\rm b} = \begin{cases} \mathrm{i}^{\rm b}_{\rm impr} \in \mathcal{T}_j^{i-1} & \text{with probability } p^{\rm b}_{\rm i} \\ \mathrm{i}^{\rm b}_{\rm rand} \in \mathcal{T}_j^{i-1} & \text{with probability } p^{\rm b}_{\rm r} \end{cases}$$
23: **end if**
24: **end for**
25: Compute function values $\Phi(\mathcal{T}_j^i)$.
26: Update the current best model $\mathcal{T}_{\rm best}$.
27: **end for**

Figure 1. Particle Swarm Stepwise Algorithm (PaSS)

PaSS contains multiple "particles", and each particle is associated with a model (i.e., a subset of the variables X_i). At the beginning of PaSS, a swarm of particles is associated with a set of different initial models. Then, at each iteration of PaSS, each of the particles updates the corresponding model via a stepwise procedure which is customized for the information-criterion-based variable selection problems. This stepwise procedure includes a forward selection step (adding new variables) or a backward elimination step (deleting variables). Furthermore, in the updating stage, each particle communicates with other particles to share their own search information to accelerate the overall convergence toward the optimal solution.

The proposed algorithm is named as PaSS because of its analogy to particle swarm optimization (PSO) which has been introduced by Kennedy and Eberhart [10]. PSO is a population-based stochastic optimization technique, whose idea is originated from simulations of social behavior, such

as bird flocks or fish schools. At the beginning of PSO, a set of particles is randomly generated in the solution space. Then the iterative procedure of PSO moves around the solution space to search optimal solution. These particles not only keep their own cognitive memories on the best values visited, but also exchange social information to obtain the best value obtained by the swarm formed by other particles. Many studies have shown its ability to provide better results in a faster and cheaper way, especially for multiple extremes and high-dimensional search regions.

While PSO is designed for continuous optimization problems originally and PaSS is designed to solve the discrete variable selection, the two algorithms share the following similarities. First, multiple particles are chosen initially. Second, each particle updates its associated data (objective function values in PSO and models in PaSS) by using information collected from itself and other particles. Despite of these similarities, we customize PaSS to solve the variable selection problems by proposing new ways to choose initial particles and to update the data associated with the particles.

We itemize the main components of PaSS below and summarize PaSS in Figure 1. The flowchart of our implementation is shown in Figure 2. Without a loss of generality, the norm of the variable X_i is assumed to be one.

A. Initialization

First, PaSS generates K particles that represent K different initial models. Each model is denoted by $\mathcal{T}_j$ for $j = 1, \ldots, K$. A model $\mathcal{T}_j$ is a subset of $\mathcal{T} = \{1, \ldots, p\}$ that represents a set of selected variables. For example, $\mathcal{T}_j = \{2, 5, 10\}$ represents the model whereby the variables X_2, X_5, and X_{10} are selected.

To generate the K initial models, we randomly selecting K different models. After generating the initial models, we compute the corresponding information criterion values and choose the model that has the minimuml information criterion value as the current best model $\mathcal{T}_{\text{best}}$.

B. Stochastic Forward-and-Backward Updating

At each PaSS iteration, each particle updates its associated model, and the corresponding objective function value is subsequently decreased. Either the *Forward Step* or the *Backward Step* is used to update the models in PaSS. In the Forward Step, we consider the following three variable adding options.

- First, we consider adding to the model the variable that results in the greatest improvement of the objective function value. Specifically, we add the variable of index $\mathrm{i}^{\mathrm{f}}_{\text{impr}}$, where
$$\mathrm{i}^{\mathrm{f}}_{\text{impr}} = \arg \max_{\mathrm{i} \in \mathcal{T} \setminus \mathcal{T}_j} |X_{\mathrm{i}}^{\top} R_{\mathcal{T}_j}|.$$
Here, $R_{\mathcal{T}_j} = Y - X_{\mathcal{T}_j} \beta_{\mathcal{T}_j}$, $X_{\mathcal{T}_j}$ is the model matrix corresponding to $\mathcal{T}_j$, and $\beta_{\mathcal{T}_j}$ is the least squares estimation of the corresponding coefficient vector. We let $\mathcal{T} = \{1, \ldots, p\}$ be the set of all feasible variable indices.
- Second, we can randomly choose an additional variable from the current best model $\mathcal{T}_{\text{best}}$. Specifically, a variable index $\mathrm{i}^{\mathrm{f}}_{\text{best}}$ is randomly selected from $\mathcal{T}_{\text{best}} \cap (\mathcal{T} \setminus \mathcal{T}_j)$. Here, $\mathcal{T}_{\text{best}}$ denotes the current best model among all K particles.
- Third, to avoid being trapped in a local minimum, we allow the model to depart from the neighborhood of a local optimal point by randomly adding the $\mathrm{i}^{\mathrm{f}}_{\text{rand}}$-th variable from $\mathcal{T} \setminus \mathcal{T}_j$.

We use a stochastic scheme to decide which of the forward options should be taken. We assume that the probability for adding the $\mathrm{i}^{\mathrm{f}}_{\text{best}}$-th variable, the $\mathrm{i}^{\mathrm{f}}_{\text{impr}}$-th variable, and the $\mathrm{i}^{\mathrm{f}}_{\text{rand}}$-th variable are $p^{\mathrm{f}}_{\mathrm{b}}$, $p^{\mathrm{f}}_{\mathrm{i}}$, and $p^{\mathrm{f}}_{\mathrm{r}}$, respectively, where $p^{\mathrm{f}}_{\mathrm{b}} + p^{\mathrm{f}}_{\mathrm{i}} + p^{\mathrm{f}}_{\mathrm{r}} = 1$. We choose one of the three forward options from a multinomial distribution with the probability vector $(p^{\mathrm{f}}_{\mathrm{b}}, p^{\mathrm{f}}_{\mathrm{i}}, p^{\mathrm{f}}_{\mathrm{r}})$.

In contrast, in the Backward Step, we consider the following two variable deleting options:

- First, the variable in the current model is removed if this action results in the least possible reduction of the model loss. Specifically, we delete the $\mathrm{i}^{\mathrm{b}}_{\text{impr}}$-th variable, where
$$\mathrm{i}^{\mathrm{b}}_{\text{impr}} = \arg \min_{\mathrm{i} \in \mathcal{T}_j} \|R_{\mathcal{T}_j \setminus \{\mathrm{i}\}}\|_2.$$
- Second, we randomly remove a variable from the current model. Specifically, we randomly select $\mathrm{i}^{\mathrm{b}}_{\text{rand}} \in \mathcal{T}_j$.

We delete the $\mathrm{i}^{\mathrm{b}}_{\text{impr}}$-th variable with probability $p^{\mathrm{b}}_{\mathrm{i}}$ and the $\mathrm{i}^{\mathrm{b}}_{\text{rand}}$-th variable with probability $p^{\mathrm{b}}_{\mathrm{r}} = 1 - p^{\mathrm{b}}_{\mathrm{i}}$. Specifically, we base the variable removal on a binomial distribution with probability $p^{\mathrm{b}}_{\mathrm{i}}$.

Finally, we check the function values $\Phi(\mathcal{T}_j)$ to decide if either a forward or backward step is taken in each iteration. For each particle, if we obtain monotonically decreasing function values with respect to the number of iterations, i.e., $\Phi(\mathcal{T}_j^{(t)}) \leq \Phi(\mathcal{T}_j^{(t-1)})$, then we keep the same updating step for the $(t+1)$-th iteration. In this case, we are on the right track. Otherwise, this result suggests that we made an incorrect updating decision in the previous iteration. Thus, we switch from a forward step to a backward step (or from a backward step to a forward step).

C. Updating $\mathcal{T}_{\text{best}}$ and Terminating the Algorithm

After updating all K models in each iteration, we first compute the corresponding K objective function values $\Phi(\mathcal{T}_j)$ for $j = 1, \ldots, K$. Then, we update the current best model $\mathcal{T}_{\text{best}}$ if necessary. Finally, when a pre-defined stopping criterion (e.g., number of total iterations) is met, the algorithm stops and reports $\mathcal{T}_{\text{best}}$ as the best model based on the pre-specified information criterion.

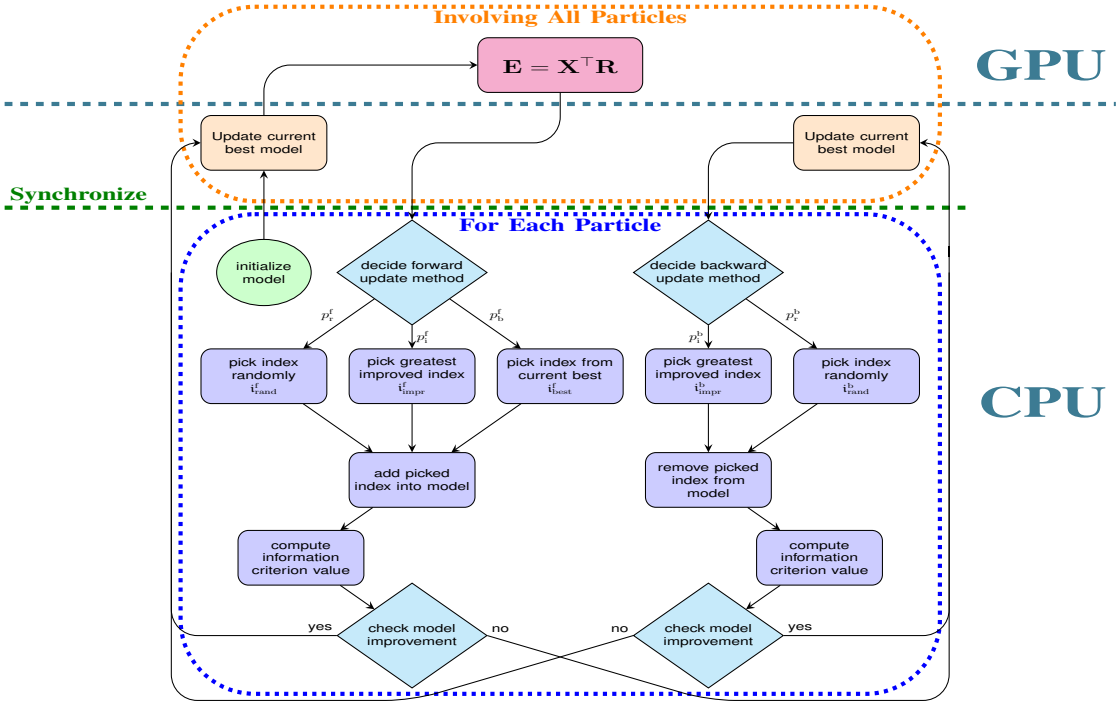

Figure 2. The flowchart of PaSS algorithm using GPU acceleration

D. Information Criteria

Several information criteria are used in our implementation:

- Akaike information criterion (AIC) [3]

$$\mathrm{AIC}(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + 2k.$$

- Bayesian information criterion (BIC) [4]

$$\mathrm{BIC}(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + k \log n.$$

- Extended Bayesian information criterion (EBIC) [5]

$$\mathrm{BIC}_\gamma(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + k \log n + 2\gamma \log \binom{p}{k}.$$

- High-dimensional Bayesian information criterion (HDBIC) [6]

$$\mathrm{HDBIC}(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + k \log n \log p.$$

- High-dimensional Hannan-Quinn information criterion (HDHQC) [6]

$$\mathrm{HDHQC}(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + 2k \log \log n \log p.$$

Here $\mathcal{T} \subseteq \{1, 2, \ldots, p\}$ is a non-empty set, k is the number of the components in $\mathcal{T}$, $\hat{\sigma}_{\mathcal{T}} = n^{-1}\|Y - \hat{Y}_{\mathcal{T}}\|_2^2$ is the 2-norm prediction error, and $0 \leq \gamma \leq 1$. AIC and BIC are designed for small size selection problems, especially for problems with $n > p$. EBIC, HDBIC, and HDHQC are introduced for large-p-small-n selection problems.

E. Parallel Acceleration

In our algorithm, the particles can be updated independently. Therefore, we use OpenMP [11] to parallel the particles in our implementation since there is no communication between particles except the step of updating the best model. Actually, we only need to update the chosen index of the best model, which is a small boolean vector with size p. Therefore, the scalability of our implementation is satisfactory as shown in Section IV-A2.

In our implementation, each thread handles several particles to improve the efficiency in forward updating step (see Section III-F). We divide the particles equally by threads. For examples, if we have τ threads and K particles, each thread will host K/τ particles. We choose to divide equally since the computation of each particle are similar.

There is no synchronization between threads in this implementation. The indices of the current best model is stored in a boolean vector. Therefore, even if copying the vector have not finished when other threads accessing this vector, we will just get the indices of a previous best model, which is also a better model. Hence, we decide not to synchronize after updating best model to avoid overhead. There will be overhead if we synchronize in each iteration since selecting greatest improved index is takes more time than selecting randomly.

Our implementation also supports multicore cluster. We use OpenMPI [12] for cross-node parallelization. Similarly, we divide the particles equally into each MPI processes. Each processes hosts K/ν particles, and each thread in a process hosts $K/(\nu\tau)$ particles for clusters with ν processes,

τ threads per process, and K particles. According to Section IV-B3, the random selection is much more important for the large-scale system. Therefore, we choose not to communicate between processes. In our implementation, each node has its best model. We only need to communicate between processes at the beginning and the end of the algorithm. That is, we send matrix $\mathbf{X}$ and vector Y to the processes at the beginning, and receive the best model at the end.

F. GPU Acceleration

We found that PaSS spends about 90% of the time on computing the inner products in forward updating step:

$$\mathrm{i}^{\mathrm{f}}_{\mathrm{impr}} = \arg \max_{\mathrm{i} \in \mathcal{T} \setminus \mathcal{T}_j} |X_{\mathrm{i}}^{\top} R_{\mathcal{T}_j}|.$$

This maximization problem can be rewritten into

$$\mathrm{i}^{\mathrm{f}}_{\mathrm{impr}} = \arg \max_{\mathrm{i} \in \mathcal{T} \setminus \mathcal{T}_j} |e_{\mathrm{i}}|,$$

where e_{i} is the i-th element of $E_j = \mathbf{X}^{\top} R_{\mathcal{T}_j}$. Since computing $\mathbf{X}^{\top} R_{\mathcal{T}_j}$ is a BLAS level 2 operation, which is faster than computing multiple BLAS level 1 operations $R_{\mathcal{T}_j}^{\top} X_{\mathrm{i}}$. Moreover, we can use a BLAS level 3 operation

$$\mathbf{E} = \mathbf{X}^{\top} \mathbf{R},$$

where $\mathbf{R} = [R_{\mathcal{T}_1}, \ldots, R_{\mathcal{T}_K}]$ and $\mathbf{E} = [E_1, \ldots, E_K]$.

Note that $\mathbf{X}$, $\mathbf{R}$, and $\mathbf{E}$ are dense matrices. Such dense matrix-matrix multiplications can be significantly accelerated by using GPU. For large-scale problems, this method is much faster than the CPU only version (see Section IV-A3).

Our implementation only supports 1 GPU per MPI process. For cluster with multiple GPUs per node, one may use multiple MPI processes on a node (with the same number of GPUs) that each rank hosts a GPU.

Figure 2 is the flowchart of our implementation. The magenta step at the top of the flowchart is in GPU. Although this method requires synchronization in each iteration, the acceleration by GPU is fast enough to ignore the disadvantage of synchronization.

We only need to send matrix $\mathbf{X}$ to the GPU once since it is stationary in the algorithm. We move $\mathbf{X}$ to GPU before initializing particles. In each iteration, we only need to send matrix $\mathbf{R}$ to GPU, and receive matrix $\mathbf{E}$ from GPU. Note that the number of particles K is much smaller than n and p. Therefore, the matrices $\mathbf{R}$ and $\mathbf{E}$ are also much smaller than $\mathbf{X}$; that is, the communication between CPU and GPU is much smaller than the computation of the multiplication. Also, since sending the data costs square of the size, and computing matrix-matrix multiplication costs about cube of the size, the larger the size is, the less the cost of communication is (respect to the computation).

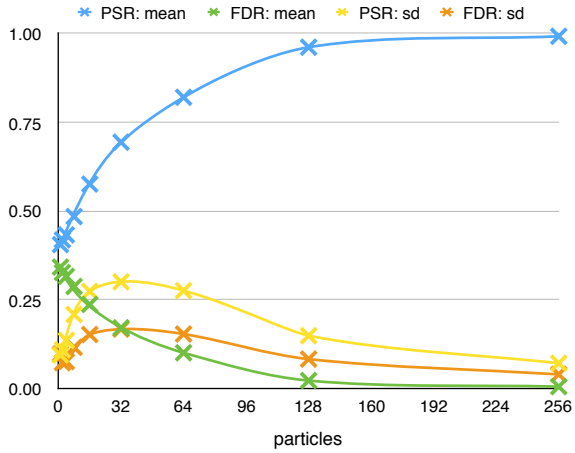

Figure 3. PSR & FPR versus the number of particles [1]

IV. Numerical Results

A. Numerical Results on Multicore Hybrid CPU-GPU Clusters

The PaSS algorithm is implemented with C++ using OpenMP [11] and OpenMPI [12] parallel for multicore clusters. Intel Math Kernel Library [13] is used for BLAS routines. The GPU version is implemented with the MAGMA library [14].

Our algorithm is implemented on IBM Cognitive Computing Cluster (Intel Xeon E5-2667 v2 @ 3.30GHz, 8 cores / 16 threads, 768GB RAM; NVIDIA K40 $\times$ 2, 12GB RAM per card) system and IBM Blue Gene/Q (IBM PowerPC A2 @ 1.6GHz, 16 cores / 64 threads) system. However, with the communication requirement on a scale of selected problem, our algorithm does not get the advantage using IBM Blue Gene Series, of which the advantage is the speed of communication. Therefore, we focus on the performance of PaSS on IBM Cognitive Computing Cluster. All of the numerical results are tested on this cluster.

1) Accuracy: We test the accuracy of PaSS using Hung et al.'s dataset 1-1 [15] (see Section IV-B3) to show the benefit of using more particles. Let S^* denote the selected variable set, where S is the real model. The table shows the positive selection rate (PSR) computed by $\#(S^* \cap S)/\#(S)$ and the false discovery rate (FDR) computed by $\#(S^* \setminus S)/\#(S^*)$. The PSR is the rate of active effects correctly identified in the measurement. The FDR is the rate of inactive effects and can be considered as the type I error rate of the selection approach. Figure 3 shows the mean and the standard deviation of PSR and FDR. It is shown that the more particles we use the more accuracy we get.

2) Scalability: The scalability of PaSS is tested using a large-p-small-n variable selection problem discussed in Ing

[1] Simulation results of data 1-1 in Hung et al.'s datasets [15] ($n = 200$ and $p = 1225$) with HDBIC using 256 iterations in 1000 tests.

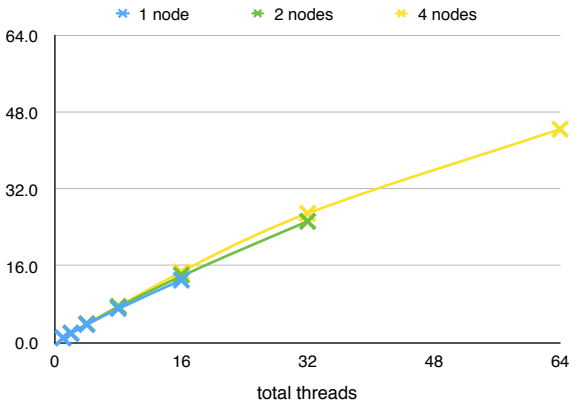

Figure 4. Speed & scalability versus the number of threads [2]

	No GPU		With GPU	
	Time (sec)	Speed	Time (sec)	Speed
Ing & Lai	117.065301	1	16.251136	7.2
Chen & Chen	97.426930	1	13.878842	7.0

Table I
GPU ACCELERATION[3]

and Lai (2011) [6] (see Section IV-B1). Figure 4 shows the scalability of this implementation using different numbers of nodes and threads. It is shown that it has near linearly scalability due to less communication between nodes (with respect to the computation).

3) GPU Acceleration: Two huge-size problems are tested on IBM Cognitive Computing Cluster, which has two K40 GPUs per node. The first problem ($n = 4000$ and $p = 40000$) uses 1,872MB memory, and the second problem ($n = 20000$ and $p = 500$) uses 2,303MB memory. Table I shows that using GPU can accelerate up to 7 times faster.

B. The Choice of Parameters

In this section, we discuss the performance of PaSS on some simulation models. We test the effects of the choice of the parameters $(p_{\rm b}^{\rm f}, p_{\rm i}^{\rm f}, p_{\rm r}^{\rm f}, p_{\rm i}^{\rm b}, p_{\rm r}^{\rm b})$ with all possible tuples of $p = 0, 0.2, \ldots, 1.0$. For the first two models, the minimum is outstanding, so that the effect backward step is inconspicuous. The last model has an outstanding local minimum so that the behavior is different.

1) Variable Selection Problem with Large-p-Small-n: First, we consider a variable selection problem discussed in Ing and Lai (2011) [6] whereby $n = 400$ and $p = 4000$.

[2]Simulation results of a large-p-small-n ($n = 400$ and $p = 4000$) variable selection problem with HDBIC considered in Ing and Lai (2011) [6] using 64 iterations and 256 particles in 100 tests. Using $1, 2, 4$ nodes and $1, 2, 4, 8, 16$ threads per node. Note that this test does not use GPU acceleration.

[3]Simulation results of two huge-size problems using GPU acceleration. One ($n = 4000$ and $p = 40000$) is a variable selection problem with HDBIC considered in Ing and Lai (2011) [6], another ($n = 20000$ and $p = 500$) is a variable selection problem with EBIC($\gamma = 1$) information criterion considered in Chen and Chen (2008) [5]. Both are tested 100 times.

16 Particles, 32 Iterations					
probability		mean		standard deviation	
$(p_{\rm b}^{\rm f}, p_{\rm i}^{\rm f}, p_{\rm r}^{\rm f})$	$(p_{\rm i}^{\rm b}, p_{\rm r}^{\rm b})$	PSR	FDR	PSR	FDR
$(0.0, 1.0, 0.0)$	$(0.8, 0.2)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(1.0, 0.0)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(0.6, 0.4)$	1.000000	0.000091	0.000000	0.002875
$(0.2, 0.8, 0.0)$	$(1.0, 0.0)$	1.000000	0.008788	0.000000	0.036801
$(0.0, 1.0, 0.0)$	$(0.4, 0.6)$	1.000000	0.012470	0.000000	0.032285
$(0.2, 0.8, 0.0)$	$(0.8, 0.2)$	1.000000	0.027258	0.000000	0.057149
$(0.0, 0.8, 0.2)$	$(0.8, 0.2)$	1.000000	0.043455	0.000000	0.068690
$(0.0, 1.0, 0.0)$	$(0.2, 0.8)$	1.000000	0.056167	0.000000	0.052916
$(0.2, 0.8, 0.0)$	$(0.6, 0.4)$	1.000000	0.061192	0.000000	0.072745

256 Particles, 16 Iterations					
probability		mean		standard deviation	
$(p_{\rm b}^{\rm f}, p_{\rm i}^{\rm f}, p_{\rm r}^{\rm f})$	$(p_{\rm i}^{\rm b}, p_{\rm r}^{\rm b})$	PSR	FDR	PSR	FDR
$(0.0, 0.8, 0.2)$	$(0.6, 0.4)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 0.8, 0.2)$	$(0.8, 0.2)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 0.8, 0.2)$	$(1.0, 0.0)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(0.2, 0.8)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(0.4, 0.6)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(0.6, 0.4)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(0.8, 0.2)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(1.0, 0.0)$	1.000000	0.000000	0.000000	0.000000
$(0.2, 0.8, 0.0)$	$(0.6, 0.4)$	1.000000	0.000000	0.000000	0.000000
$(0.2, 0.8, 0.0)$	$(0.8, 0.2)$	1.000000	0.000000	0.000000	0.000000
$(0.2, 0.8, 0.0)$	$(1.0, 0.0)$	1.000000	0.000000	0.000000	0.000000
$(0.4, 0.6, 0.0)$	$(1.0, 0.0)$	1.000000	0.000000	0.000000	0.000000
$(0.2, 0.8, 0.0)$	$(0.4, 0.6)$	1.000000	0.000091	0.000000	0.002875
$(0.4, 0.6, 0.0)$	$(0.8, 0.2)$	1.000000	0.000788	0.000000	0.009816

Table II
SIMULATION RESULTS OF ING & LAI'S MODEL [4]

In this problem, $X_1, \ldots, X_{\rm r}$ ($r = 10$) are generated independently from a multivariate normal distribution with zero mean vector and covariance matrix I_n. For $X_i = (x_{1i}, \ldots, x_{ni})^\top$ that $i > r$, we have

$$x_{t,i} = d_{t,i} + b \sum_{l=1}^{r} x_{t,l}, \qquad r < i \le p,\ t = 1, \ldots, n.$$

Here, $b = (3/4r)^{1/2}$ and $(d_{t,(r+1)}, \ldots, d_{t,p})^\top$ are i.i.d. multivariate normal with zero mean and covariance matrix $\frac{1}{4} I_{(p-r)}$, and they are independent of x_{ti} for $1 \le j \le r$. The non-zero coefficient vector is $(\beta_1, \ldots, \beta_{\rm r}) = (3, 3.75, \ldots, 9.75)$. For all $i > r$, we have $\beta_i = 0$. The response Y is generated from (1) with the noise vector ϵ drawn from the multivariate normal distribution with zero mean vector and covariance matrix I_n. In this example, HDBIC

$$\mathrm{HDBIC}(\mathcal{T}) = n \log \hat{\sigma}_{\mathcal{T}}^2 + \#(\mathcal{T}) \log n \log p$$

is used as the selection criterion.

Table II shows the performance of the best choices of the parameters on the data with different numbers of particles and iterations. We recommend high $p_{\rm i}^{\rm f}, p_{\rm i}^{\rm b}$ and low $p_{\rm b}^{\rm f}, p_{\rm r}^{\rm f}, p_{\rm r}^{\rm b}$ to reach the minimum quickly. Also, it is shown that using more particles has better accuracy with less number of iterations.

16 Particles, 8 Iterations					
probability		mean		standard deviation	
$(p_{\mathrm{b}}^{\mathrm{f}}, p_{\mathrm{i}}^{\mathrm{f}}, p_{\mathrm{r}}^{\mathrm{f}})$	$(p_{\mathrm{i}}^{\mathrm{b}}, p_{\mathrm{r}}^{\mathrm{b}})$	PSR	FDR	PSR	FDR
$(0.0, 1.0, 0.0)$	$(0.2, 0.8)$	0.759500	0.008554	0.033141	0.033878
$(0.0, 1.0, 0.0)$	$(0.8, 0.2)$	0.757500	0.010107	0.029701	0.036587
$(0.0, 1.0, 0.0)$	$(1.0, 0.0)$	0.757875	0.010143	0.030386	0.036708
$(0.0, 1.0, 0.0)$	$(0.0, 1.0)$	0.758250	0.010286	0.031051	0.036945
$(0.0, 1.0, 0.0)$	$(0.4, 0.6)$	0.757500	0.010554	0.029701	0.037356
$(0.0, 1.0, 0.0)$	$(0.6, 0.4)$	0.757000	0.010554	0.028755	0.037356
$(0.2, 0.8, 0.0)$	$(0.2, 0.8)$	0.758875	0.011048	0.036667	0.039002
$(0.2, 0.8, 0.0)$	$(0.8, 0.2)$	0.754250	0.011268	0.022665	0.038496
$(0.2, 0.8, 0.0)$	$(1.0, 0.0)$	0.759625	0.011696	0.033341	0.039159
$(0.2, 0.8, 0.0)$	$(0.0, 1.0)$	0.759250	0.013268	0.032738	0.041459

256 Particles, 8 Iterations					
probability		mean		standard deviation	
$(p_{\mathrm{b}}^{\mathrm{f}}, p_{\mathrm{i}}^{\mathrm{f}}, p_{\mathrm{r}}^{\mathrm{f}})$	$(p_{\mathrm{i}}^{\mathrm{b}}, p_{\mathrm{r}}^{\mathrm{b}})$	PSR	FDR	PSR	FDR
$(0.0, 0.8, 0.2)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.2, 0.6, 0.2)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.2, 0.8, 0.0)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.4, 0.6, 0.0)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.6, 0.4, 0.0)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.6, 0.4, 0.0)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.8, 0.2, 0.0)$	$*$	0.750000	0.000000	0.000000	0.000000
$(0.8, 0.2, 0.0)$	$(0.2, 0.8)$	0.750125	0.000000	0.003953	0.000000
$(0.4, 0.4, 0.2)$	$(0.4, 0.6)$	0.750250	0.000000	0.005587	0.000000

Table III
SIMULATION RESULTS OF CHEN & CHEN'S MODEL [5]

16 Particles, 256 Iterations					
probability		mean		standard deviation	
$(p_{\mathrm{b}}^{\mathrm{f}}, p_{\mathrm{i}}^{\mathrm{f}}, p_{\mathrm{r}}^{\mathrm{f}})$	$(p_{\mathrm{i}}^{\mathrm{b}}, p_{\mathrm{r}}^{\mathrm{b}})$	PSR	FDR	PSR	FDR
$(0.2, 0.6, 0.2)$	$(0.2, 0.8)$	1.000000	0.000000	0.000000	0.000000
$(0.0, 1.0, 0.0)$	$(0.2, 0.8)$	0.999600	0.002911	0.012649	0.024703
$(0.0, 0.8, 0.2)$	$(0.4, 0.6)$	0.999600	0.004083	0.012649	0.030393
$(0.0, 0.8, 0.2)$	$(0.2, 0.8)$	0.999200	0.015002	0.015479	0.056618
$(0.2, 0.6, 0.2)$	$(0.4, 0.6)$	0.998400	0.001619	0.025260	0.019620
$(0.0, 1.0, 0.0)$	$(0.4, 0.6)$	0.998200	0.002577	0.026028	0.030693
$(0.2, 0.8, 0.0)$	$(0.2, 0.8)$	0.996800	0.002893	0.035652	0.030329
$(0.2, 0.6, 0.2)$	$(0.0, 1.0)$	0.996800	0.016467	0.033330	0.056010
$(0.0, 0.8, 0.2)$	$(0.0, 1.0)$	0.995800	0.145713	0.034980	0.124730
$(0.2, 0.8, 0.0)$	$(0.4, 0.6)$	0.995200	0.003766	0.043576	0.034620

256 Particles, 32 Iterations					
probability		mean		standard deviation	
$(p_{\mathrm{b}}^{\mathrm{f}}, p_{\mathrm{i}}^{\mathrm{f}}, p_{\mathrm{r}}^{\mathrm{f}})$	$(p_{\mathrm{i}}^{\mathrm{b}}, p_{\mathrm{r}}^{\mathrm{b}})$	PSR	FDR	PSR	FDR
$(0.2, 0.8, 0.0)$	$(0.2, 0.8)$	1.000000	0.000000	0.000000	0.000000
$(0.2, 0.8, 0.0)$	$(0.4, 0.6)$	1.000000	0.000000	0.000000	0.000000
$(0.4, 0.6, 0.0)$	$(0.2, 0.8)$	1.000000	0.000167	0.000000	0.005270
$(0.4, 0.6, 0.0)$	$(0.0, 1.0)$	1.000000	0.000667	0.000000	0.010525
$(0.2, 0.8, 0.0)$	$(0.0, 1.0)$	1.000000	0.001119	0.000000	0.014815
$(0.0, 1.0, 0.0)$	$(0.4, 0.6)$	1.000000	0.020843	0.000000	0.059873
$(0.0, 1.0, 0.0)$	$(0.2, 0.8)$	0.999800	0.049595	0.006325	0.082176
$(0.4, 0.6, 0.0)$	$(0.4, 0.6)$	0.999600	0.001155	0.012649	0.016741
$(0.2, 0.6, 0.2)$	$(0.2, 0.8)$	0.999600	0.002077	0.012649	0.021803
$(0.0, 1.0, 0.0)$	$(0.0, 1.0)$	0.999400	0.148825	0.010943	0.100355

Table IV
SIMULATION RESULTS OF HUNG ET AL.'S DATA 1-1 [6]

2) Variable Selection Problem with Large-n-Small-p: The second example is taken from Chen and Chen (2008) [5] with $n = 200$ and $p = 50$. In this example, the components of X_i are generated from $N(0, 1)$ (normal distribution), and the correlation structure among variables is defined as

$$\mathrm{Cov}(X_i, X_j) = \rho^{|i-j|}$$

for all i, j, where we consider $\rho = 0.2$ in this example. The non-zero coefficient vector is set as $(\beta_1, \ldots, \beta_8) = (0.7, 0.9, 0.4, 0.3, 1.0, 0.2, 0.2, 0.1)$. The response Y is generated by (1) and the noise vector ϵ comes from a multivariate normal distribution with a zero mean vector and the covariance matrix I_n. Here we use EBIC

$$\mathrm{BIC}_\gamma(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + \#(\mathcal{T}) \log n + 2\gamma \log \binom{p}{\#(\mathcal{T})}$$

as the selection criterion.

Table III shows the performance of the best choices of the parameters on the data with different numbers of particles and iterations. We found that high $p_{\mathrm{i}}^{\mathrm{f}}$ is recommended to reach the minimum quickly, and the effects of $p_{\mathrm{i}}^{\mathrm{b}}$ and $p_{\mathrm{r}}^{\mathrm{b}}$ are inconspicuous since the minimum is outstanding. Although using more particles does not increase the accuracy, it makes the result more stable.

[4]Simulation results of a large-p-small-n ($n = 400$ and $p = 4000$) variable selection problem with HDBIC considered in Ing and Lai (2011) [6] in 1000 tests.

[5]Simulation results of a large-n-small-p ($n = 200$ and $p = 50$) variable selection problem with EBIC($\gamma = 1$) information criterion considered in Chen and Chen (2008) [5] in 1000 tests. The $*$ in the second table mean that the results are the same for all pairs of $(p_{\mathrm{i}}^{\mathrm{b}}, p_{\mathrm{r}}^{\mathrm{b}})$.

3) Variable Selection Problem with Outstanding Local Best: Hung et al.'s dataset [15] is a good example to show the importance of random selection. The datasets are from the multi-locus models

$$Y = \gamma + \sum_{i=1}^{p} \xi_i g_i + \sum_{i<j} \eta_{ij} \cdot (g_i g_j) + \epsilon,$$

where ξ_i is the main effect of the i-th locus, and η_{ij}, $i < j$, is corresponds to the interaction of i-th and j-th loci. In each simulated dataset are generated with genotype and trait values of 400 individuals. For genotypes, 1000 SNPs $G = (g_1, \ldots, g_{1000})^\top$ with $g_i \in \{0, 1, 2\}$ are generated, where marginally each g_i is obtained from a discretization of a standard normal distribution with $P(g_i = 0) = P(g_i = 2) = \frac{1}{4}$ and $P(g_i = 1) = \frac{1}{2}$. The 1000 SNPs can be grouped into 200 blocks of 5 SNPs, where SNPs from different blocks are independent and SNPs within the same block are correlated with an R-square of 0.3^2. Y is generated by

$$Y = \beta \cdot (g_5 g_6 + 0.8 g_{10} g_{11} + 0.6 g_{15} g_{16} + 2 g_{20} + 2 g_{21}) + \epsilon,$$

where β is the effort size and ϵ from normal distribution with zero mean vector and covariance matrix I_n. In this example, we use HDHQC [6]

$$\mathrm{HDHQC}(\mathcal{T}) = n \log \hat{\sigma}^2_{\mathcal{T}} + 2\#(\mathcal{T}) \log \log n \log p$$

as the selection criterion.

[6]Simulation results of data 1-1 in Hung et al.'s datasets [15] ($n = 200$ and $p = 1225$) with HDHQC in 1000 tests.

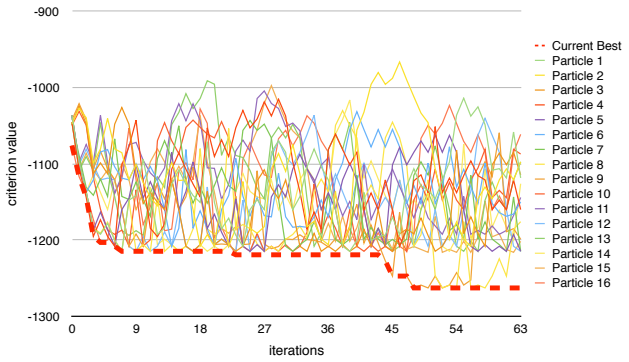

Figure 5. The criterion value versus the number of iterations [7]

Figure 5 shows the criterion value of each particle and the best model (the red thick dash line). The criterion value of the real model is about -1263. However, it is shown that almost all particles reach a local minimum with $\Phi = -1215$. After reaching the local minimum, local best selections ($\mathrm{i}^{\mathrm{f}}_{\mathrm{impr}}$ and $\mathrm{i}^{\mathrm{b}}_{\mathrm{impr}}$) are useless, and the algorithm needs to wait for a random jump to escape the trap of the local minimum to search for the global minimum.

Table IV shows the performance of the best choices of the parameters on the data with different numbers of particles and iterations. We observed that high $p^{\mathrm{b}}_{\mathrm{r}}$ is required for jumping out of the local minimum to reach the global minimum point, and high $p^{\mathrm{f}}_{\mathrm{i}}$ is needed to reach a minimum quickly.

V. Conclusion

Particle Swarm Stepwise Algorithm (PaSS) has been proposed to solve variable selection problems via information criterion optimization. PaSS simultaneously explores multiple models and shares the search information associated with all explored models. This algorithm also uses a new stochastic stepwise scheme to search the best model.

We have shown that PaSS can be efficiently implemented on parallel clusters with multiple threaded CPU and many-core GPU. Multithread and multicore parallelization are used to speed up the computation. Communication is reduced for better scalability. The speedup due to the parallelism can significantly reduce the runtime. For example, by using 2, 4, 8, 16, 32 and 64 threads, we can achieve about 1.99, 3.96, 7.58, 14.7, 26.8 and 44.4 times faster comparing with the performance of the single thread, respectively. In addition, by using a NVIDIA K40 GPU to accelerate the computation of a huge matrix-matrix multiplication, we can realize an extra 7X faster performance gain in timing. This fast parallel PaSS solver can solve the information-criterion-based variable selection problems with big datasets. Consequently, we can use it to tackle bigger problems in data mining, genomes analysis, machine learning, and numerical simulations. Based on our numerical experience, we suggest using large $\mathrm{i}^{\mathrm{f}}_{\mathrm{impr}}$-th variable ($p^{\mathrm{f}}_{\mathrm{i}}$) and large $\mathrm{i}^{\mathrm{f}}_{\mathrm{rand}}$-th variable ($p^{\mathrm{b}}_{\mathrm{r}}$) for an information criterion selection approach on large scale systems.

[7] Simulation results of data 1-1 in Hung et al.'s datasets [15] ($n = 200$ and $p = 1225$) with HDHQC using 16 particles.

References

[1] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[2] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM review*, vol. 43, no. 1, pp. 129–159, 2001.

[3] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected Papers of Hirotugu Akaike*. Springer, 1998, pp. 199–213.

[4] G. Schwarz *et al.*, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[5] J. Chen and Z. Chen, "Extended bayesian information criteria for model selection with large model spaces," *Biometrika*, vol. 95, no. 3, pp. 759–771, 2008.

[6] C.-K. Ing and T. L. Lai, "A stepwise regression method and consistent model selection for high-dimensional sparse linear models," *Statistica Sinica*, pp. 1473–1513, 2011.

[7] J. Fan and R. Li, "Variable selection via nonconcave penalized likelihood and its oracle properties," *Journal of the American statistical Association*, vol. 96, no. 456, pp. 1348–1360, 2001.

[8] T. Zhang, "Adaptive forward-backward greedy algorithm for learning sparse representations," *Information Theory, IEEE Transactions on*, vol. 57, no. 7, pp. 4689–4708, 2011.

[9] R.-B. Chen, C.-C. Huang, and W. Wang, "Particle swarm stepwise (PaSS) algorithm for vairable selection," 2015, technical report.

[10] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766.

[11] L. Dagum and R. Enon, "Openmp: an industry standard api for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[12] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2004, pp. 97–104.

[13] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, "Intel math kernel library," in *High-Performance Computing on the Intel® Xeon Phi*. Springer, 2014, pp. 167–188.

[14] S. Tomov, J. Dongarra, V. Volkov, and J. Demmel, "Magma library," *Univ. of Tennessee and Univ. of California, Knoxville, TN, and Berkeley, CA*, 2009.

[15] H. Hung, Y.-T. Lin, P. Chen, C.-C. Wang, S.-Y. Huang, and J.-Y. Tzeng, "Detection of gene–gene interactions using multistage sparse and low-rank regression," *Biometrics*, 2015.