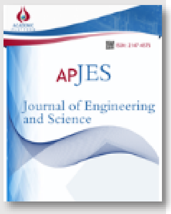
Academic Platform Journal of Engineering and Science

journal homepage: http://apjes.com/



Performance Analysis of Machine Learning and Bioinformatics Applications on High Performance Computing Systems

*[1]Zafer Aydın
[1]Abdullah Gül Üniversity, Department of Computer Engineering, Kocasinan, Kayseri, Turkey,
zafer.aydin@agu.edu.tr,

Research Paper   Arrival Date: 30.03.2019   Accepted Date: 28.08.2019

Abstract

Nowadays, it is becoming increasingly important to use the most efficient and most suitable computational resources for algorithmic tools that extract meaningful information from big data and make smart decisions. In this paper, a comparative analysis is provided for performance measurements of various machine learning and bioinformatics software including scikit-learn, Tensorflow, WEKA, libSVM, ThunderSVM, GMTK, PSI-BLAST, and HHblits with big data applications on different high performance computer systems and workstations. The programs are executed in a wide range of conditions such as single-core central processing unit (CPU), multi-core CPU, and graphical processing unit (GPU) depending on the availability of implementation. The optimum number of CPU cores are obtained for selected software. It is found that the running times depend on many factors including the CPU/GPU version, available RAM, the number of CPU cores allocated, and the algorithm used. If parallel implementations are available for a given software, the best running times are typically obtained by GPU, followed by multi-core CPU, and single-core CPU. Though there is no best system that performs better than others in all applications studied, it is anticipated that the results obtained will help researchers and practitioners to select the most appropriate computational resources for their machine learning and bioinformatics projects.

Keywords: Machine learning, bioinformatics, high performance computing, speed performance analysis

1. INTRODUCTION

We live in an age with millions of data generated every day. Originally introduced in the fields of astronomy, genomics, and bioinformatics, the concept of big data is now showing itself in every aspect of our lives. The Internet search engine Google presents huge amounts of data in every field, from diagnostics and treatment of diseases to shopping on the internet. In order to understand the information hidden in large collections of data, various computational tools, technologies and disciplines are brought together such as databases, computer programming, algorithms, high performance computing, data mining, machine learning, and artificial intelligence. As a result of this endeavor, customer trends and preferences can be envisaged dynamically, companies can make better strategic and innovative decisions, weather, economic trends, energy and service demands can be forecasted better to plan resources more effectively. Furthermore, risk factors, security threads can be predicted ahead of time, diseases can be prognosed and prevented at an early stage, new drugs and therapies can be developed, all of which improve the quality of our lives.

Discovering and processing information in large collections of data calls for intelligent and efficient algorithms. For this purpose, artificial intelligence and machine learning methods are widely employed. Example applications of these techniques include object recognition from images, speech recognition, natural language processing, video classification, recommender systems, anomaly detection, forecasting, disease detection, survival analysis, churn prediction, and analyzing genes/proteins of organisms. Various algorithms have been proposed in the literature for making smart decisions. In addition to selecting the most suitable algorithm for the application of interest, it is also important to choose the right computational resources (both in terms of hardware and software) in order to process information accurately and efficiently. The running time of an algorithm can be different from one software to another. Furthermore, the possibility of executing the algorithm simulatenously on multiple processors in parallel (e.g. multi-core CPUs or GPUs) may improve the running time of an algorithm dramatically [1]. Based on this fact, high performance computing (HPC) systems have been developed that contain parallel file systems, fast communication interfaces, CPU and GPU nodes, all of

Corresponding Author: Abdullah Gül Üniversity, Department of Computer Engineering, Kocasinan, Kayseri, Turkey, zafer.aydin@agu.edu.tr

Doi: 10.21541/apjes.547016

which aim to maximize the execution speed of computer programs [2].

There has been a number of studies in the literature that analyze the performance of different machine learning software. Kochura et al. compared Tensorflow, DeepLearning4J, and H2O in single and multi threaded modes [3]. Kovalev et al. compared Theano, Torch, Caffe, Tensorflow, Keras, and DeepLearning4J in terms of speed and accuracy [4]. Shatnawi et al. compared Tensorflow, Keras, Theano, and Microsoft's CNTK in terms of performance for object recognition from images [5]. Bahrampour et al. compared Caffe, Neon, Tensorflow, Theano, and Torch in terms of extensibility, hardware utilization and speed for implementing convolutional and recurrent neural networks [6]. In addition to these, there are also studies that compare the performance of software developed for other big data applications such as bioinformatics. Among those, Bader et al. developed a benchmark suite to evaluate the performance of bioinformatics software on HPC architectures [7] and Kurtz et al. compared performance of bioinformatics software on multi-core systems [8].

To date, most of the work in the literature concentrated on analyzing and comparing the speed of software only. Despite the prevalent work on popular software on deep learning and artificial intelligence, there is not much work that analyzes how these software behave in different HPC systems and how they compare to standard workstations. This could be important because running a progam on an HPC cluster may not always be the best option due to waiting times in the queues. Furthermore, the available hardware resources in an HPC may not be the best choices for a given computer software.

This paper focuses on analyzing the speed of software and algorithms in selected fields of machine learning and bioinformatics on various HPC systems in Turkey, on workstations and on single-core CPU, multi-core CPU, and GPU platforms including NVIDIA's recently developed DGX-1, which is one of the fastest system architectures specialized for artificial intelligence [9]. The following scientific problems are considered: protein sequence alignment, protein structure prediction, respiratory viral infection detection, and optical character recognition by deep learning, all of which originate from scientific and/or technological projects. To the best of our knowledge there is no work in the literature that compares the performance of the selected software related to these problems on different HPC systems. The machine learning software tested include Tensorflow [10] scikit-learn [11], WEKA [12], Graphical Models Toolkit (GMTK) [13], libSVM [14], and ThunderSVM [15] for optical character recognition, protein structure prediction, and respiratory virus infection detection problems in which the train sets contain thousands of examples and hundreds of features. Among bioinformatics software, PSI-BLAST [16] and HHblits [17] are employed to align a protein's amino acid sequence to sequences in a large database with millions of proteins. All of these applications require processing millions or billions of data elements.

2. MATERIAL AND METHOD

In the subsequent sections, the software and computer systems used in this study will be explained in more detail.

2.1. Operating System, Software and Algorithms

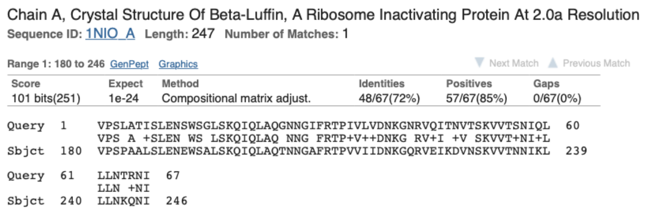
This section details the computational methods and algorithms implemented for bioinformatics, health informatics, and machine learning problems along with the software used.

2.1.1. Operating System

All of the methods are implemented and tested using a Linux operating system with Ubuntu as the Linux distribution.

2.1.2. Protein Sequence Alignment by PSI-BLAST

Protein sequence alignment is one of the widely used applications in bioinformatics research projects. The amino acid sequence of a query protein whose structure and/or function is unknown is compared against the sequences of database proteins. This enables to understand (i.e. annotate) the structure and/or function of the query protein by finding matches against database proteins with known function. PSI-BLAST [16] is a popular algorithm part of the BLAST software that is used to align the amino acid sequence of a query protein with millions of proteins in the non-redundant protein database (NR) of National Center for Biotechnology Information (NCBI) [18]. It is an iterative algorithm, which is developed mainly for finding hits (i.e. subject proteins) that have similar amino acid sequence as the query as well as distant hits that have low sequence similarity but high structural and/or functional similarity with the query. An example PSI-BLAST alignment is shown in Figure 1.


Chain A, Crystal Structure Of Beta-Luffin, A Ribosome Inactivating Protein At 2.0a Resolution
Sequence ID: 1NIO_A Length: 247 Number of Matches: 1

Range 1: 180 to 246 GenPept Graphics ▼ Next Match ▲ Previous Match

Score	Expect	Method	Identities	Positives	Gaps
101 bits(251)	1e-24	Compositional matrix adjust.	48/67(72%)	57/67(85%)	0/67(0%)

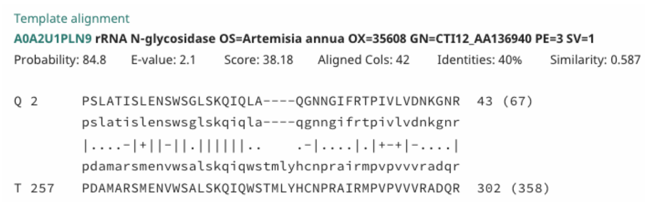
```
Query  1    VPSLATISLENSWSGLSKQIQLAQGNNGIFRTPIVLVDNKGNRVQITNVTSKVVTSNIQL  60
            VPS A +SLEN WS LSKQIQLAQ NNG FRTP+V++DNKG RV+I +V SKVVT+NI+L
Sbjct  180  VPSPAALSLENEWSALSKQIQLAQTNNGAFRTPVVIIDNKGQRVEIKDVNSKVVTNNIKL  239

Query  61   LLNTRNI  67
            LLN +NI
Sbjct  240  LLNKQNI  246
```

Figure 1. PSI-BLAST alignment between two proteins

In this paper, PSI-BLAST version 2.7.1 is executed on various computing systems for flavodoxin protein with Protein Data Bank (PDB) ID 1FX1A, which contains 147 amino acids [19]. To compute these alignments, the NR database dated as September 4, 2018 is employed that contains 167,895,434 amino acid sequences and 61,228,200,318 amino acids. The following parameters are used for PSI-BLAST: number of iterations=3, e-value threshold=10, inclusion threshold=0.001, and the number of threads=28. Therefore the parallel version of PSI-BLAST is executed on multi-core CPUs for speed comparison. Note that each query protein can have a different number of amino acids and therefore the running time of PSI-BLAST can be different for each protein.

2.1.3. Protein Sequence Alignment by HHblits

Formerly known as HHsearch, HHblits is an iterative protein sequence alignment algorithm that employs hidden Markov model (HMM) profiles and is capable of finding proteins with high structural and/or functional similarity to the query protein [17]. The sequence similarity between query and hits may be low or high depending on the availability of distant or close matches. HHblits is shown to provide more sensitive alignments and can find more distant hits than PSI-BLAST. An example HHblits alignment is shown in Figure 2.



Template alignment

A0A2U1PLN9 rRNA N-glycosidase OS=Artemisia annua OX=35608 GN=CTI12_AA136940 PE=3 SV=1

Probability: 84.8 E-value: 2.1 Score: 38.18 Aligned Cols: 42 Identities: 40% Similarity: 0.587

```
Q 2      PSLATISLENSWSGLSKQIQLA----QGNNGIFRTPIVLVDNKGNR  43 (67)
         pslatislenswsglskqiqla----qgnngifrtpivlvdnkgnr
         |....-|+||-||.||||||..    .-|....|.|+-+|-....|
         pdamarsmenvwsalskqiqwstmlyhcnprairmpvpvvvradqr
T 257    PDAMARSMENVWSALSKQIQWSTMLYHCNPRAIRMPVPVVVRADQR  302 (358)
```

Figure 2. HHblits alignment between two proteins

In this paper, the query protein 1FX1A is aligned with the sequence database using the hhblits utility and the multiple alignment is computed using the hhmake utility of HHblits. Uniprot20 is used as the sequence database dated as February 2016, which contains 8,290,068 proteins, 1,874,100,330 amino acids, and PDB70 as the HMM-profile database dated as 06 September 2014, which contains 36,595 proteins and 9,303,025 amino acids. The following parameters are used when computing the alignments: the number of iterations in the first step=2, the number of iterations in the second step=1, the number of threads=28. After performing the first step and building an HMM-profile, secondary structure sequence of the query is predicted using PSIPRED version 2.6 [20] and added to the HMM-profile model using addss.pl script of HHblits. The HMM-profile of the query is aligned with the HMM-profiles in the PDB70 database using the hhblits utility. During this step, the secondary structure label sequences of the hits are obtained from the DSSP database [21] automatically by HHblits.

2.1.4. Random forest by scikit-learn

Scikit-learn is a Python [22] library for implementing machine learning methods [11]. To analyze the performance of scikit-learn, a random forest classifier is implemented, which combines predictions from multiple decision trees using the bagging ensemble technique [23]. A random forest model is depicted in Figure 3.

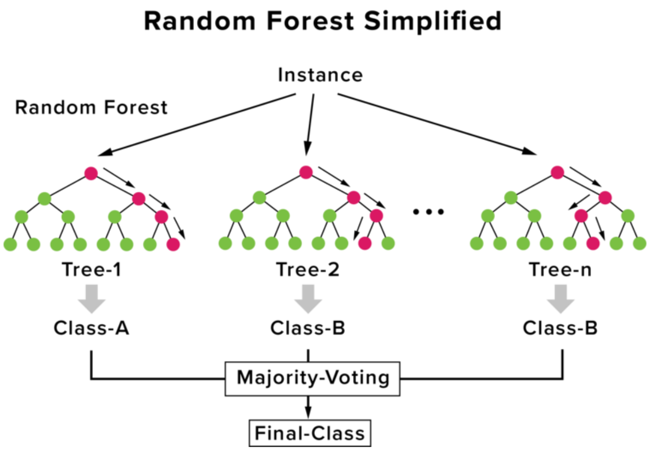


Figure 3. A random forest classification model [24]

A machine learning classifier aims to find a mapping between a given input feature vector $\mathbf{x} = [x_1, x_2, \dots, x_D]$ and an output variable y so that

$$y = f(\mathbf{x}) \qquad (1)$$

where x_i $(1 \leq i \leq D)$ represent feature variables (i.e. parameters), D is the number of dimensions and y can take a discrete class value. For example, each tree model in Figure 3 is a decision tree classifier that produce a class label y $(1 \leq y \leq K)$ as the output signal where K is the number of possible class types. Then the random forest model chooses the particular class type that is predicted most frequently by the decision tree classifiers according to the majority voting rule.

In order to train a machine learning classifier, a set of training samples are used, which are denoted as $\mathbf{x}_n$ $(1 \leq n \leq N)$ where N is the number of samples in training set. Similarly, to evaluate the prediction accuracy of a classifier, a set of test examples are used. Training in this regard corresponds to learning the function $f(.)$ from the training set by minimizing a cost function with the ultimate goal of making correct predictions for examples that are in training set and those that are outside (i.e. for new samples).

For the random forest classifier model, a train set (with 1,000,000 samples and 100 features) and a test set (with 100,000 samples and 100 features) are generated artificially using the make_classification method of scikit-learn's dataset class. The number of class labels is set to 2 (i.e. a binary classification problem). Then a random forest model is trained on the train set and class predictions are computed on test set. The number of trees parameter is set to 100, max_depth to 2, random_state to 0, and n_jobs to 1 and 28 (i.e. single-core and multi-core executions). The scikit-learn version 0.20.2 is used in TRUBA, version 0.19.1 in İTÜ Uhem, version 0.20.1 in AGÜ HPC and version 0.19.2 is in all the remaining systems. These versions, albeit slightly different, are close to each other, which will not affect the performance of random forest models significantly. For instance, though the scikit-learn version in TRUBA is newer than the other systems, it did not provide the fastest execution times as demonstrated in the results section. For this reason, the performance differences can be largely attributed to the hardware resources instead of the software versions.

2.1.5. Multi-layer perceptron by WEKA

WEKA software is developed in Java programming language for implementing various machine-learning methods [12]. In this paper, a multi-layer perceptron (MLP) neural network model [25] is implemented using WEKA for respiratory virus infection detection. An MLP architecture is illustrated in Figure 4.

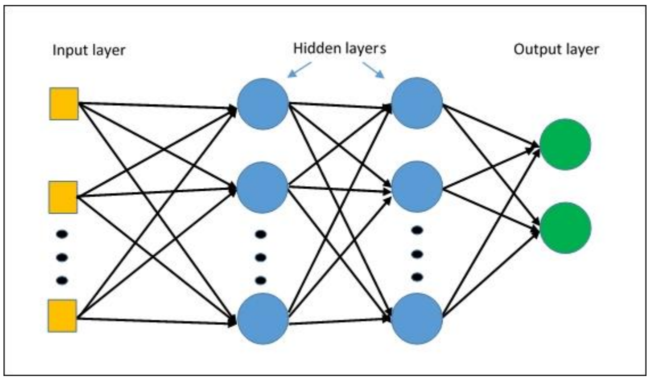


Figure 4. A multi-layer perceptron neural network model [26]

In this architecture, the yellow squares represent input feature values x_i $(1 \leq i \leq D)$, the blue circles represent hidden nodes $h_j^{(l)}$ $(1 \leq l \leq 2)$, $(1 \leq j \leq M)$, the green circles represent the output nodes y_k $(1 \leq k \leq 2)$. The output signal is computed by propagating the input feature vector $\mathbf{x} = [x_1, x_2, \dots, x_D]$ from left to right in the network. The output signal at the hidden nodes (i.e. the one on the left) is computed by the following relation

$$h_j^{(1)} = \varphi(a_j) \tag{2}$$

where $h_j^{(1)}$ represents the output of the j^{th} hidden node of the first hidden layer, $\varphi(.)$ denotes a non-linear activation function and a_j is computed as

$$a_j = \sum_{i=1}^{D} w_{\boldsymbol{ij}}^{(1)} x_{\boldsymbol{i}} \tag{3}$$

where $w_{\boldsymbol{ij}}^{(1)}$ is the weight parameter between the i^{th} input feature and the j^{th} hidden node of the first hidden layer. Applying the same functions for the second hidden layer, output of the network can be obtained as

$$\mathbf{y} = f(\mathbf{x}, \mathbf{w}) \tag{4}$$

where $\mathbf{y} = [y_1, y_2, \dots, y_K]$ with K being the number of possible class types and $\mathbf{w}$ is a vector that contains the set of all weight parameters of the network.
Based on this formulation, training a neural network corresponds to learning a set of weight parameters $\mathbf{w}$ that minimize a cost function. If squared loss is selected as the cost metric then the goal of training becomes finding $\mathbf{w}$ that minimize

$$E(\mathbf{w}) = \sum_{n=1}^{N} (\mathbf{y}_n - \mathbf{t}_n)^2 \tag{5}$$

where $\mathbf{y}_n$ is the network's output for the n^{th} data sample and $\mathbf{t}_n$ is the true output for the n^{th} data sample.

The data set used for training and testing the MLP model contained gene expression data for Respiratory Viral DREAM Challenge, which was an international competition held in 2016-2017 and organized by Sage Bionetworks, Duke University, and Darpa [27]. The goal in this challenge was to predict whether a person will be infected by respiratory flu viruses (before and after being exposed to virus). Data is obtained by performing microarray experiments using Human Affymetrix assay and blood samples of the subjects [27]. The dataset contained 22,276 gene features, 118 samples (i.e. subjects), and 2 class labels, which represents whether a flu virus will be present in nasal samples of the subjects. The same dataset is used both for training and testing phases of the MLP model. The number of hidden layers is set to 1, the number of hidden units to 5, random number seed to 0, and number of threads to 1 and 28 (i.e. both single-core and multi-core CPU platforms are considered). Model training is performed by conjugate gradient algorithm [28], which is recommended when the number of features is large. In all experiments, WEKA version 3.8.0 is employed.

2.1.6. Deep Convolutional Neural Network by Tensorflow

Tensorflow is a Python library developed by Google for implementing neural network models [10]. In this paper, a deep convolutional network model [29] is implemented using Tensorflow for optical character recognition [30] from images. An example convolutional network is shown in Figure 5.

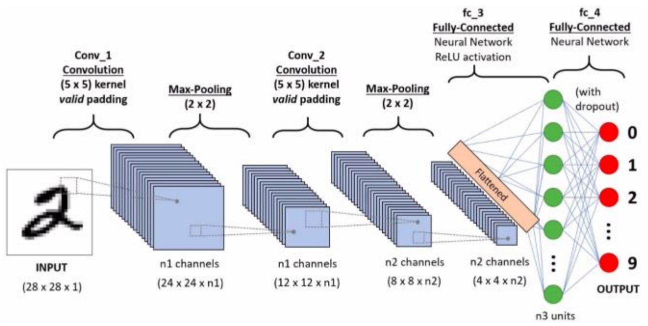


Figure 5. A deep convolutional neural network for image recognition [31]

A convolutional neural network operates similar to a multi-layer perceptron network. Instead of having a fully connected architecture it has a sparse structure with most of the weights are set to zero. In this paper, notMNIST dataset [32] is used which contains 28 by 28 images of characters ranging from A to J as shown in Figure 6. This dataset is more challenging than the standard MNIST dataset [33], which is used for recognizing digits.

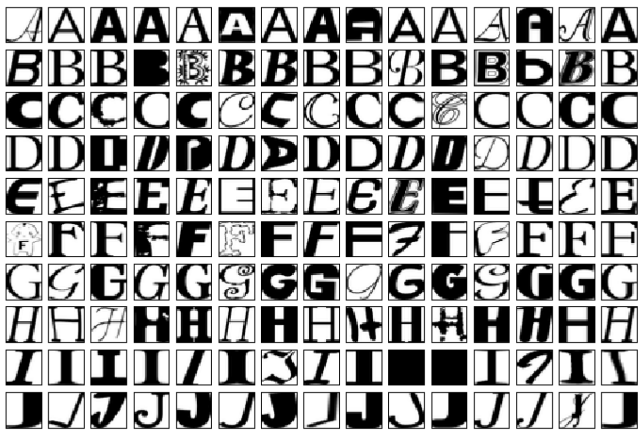


Figure 6. notMNIST dataset of characters from A to J [34]

The notMNIST dataset is randomly partitioned into a train set of 200,000 images and a test set with 10,000 images. The convolutional network contains one input layer, three convolutional layers, two fully connected MLP layers and an output layer. The network has the following specifications. A 5 by 5 kernel is used at the convolutional layers. The “same” zero padding strategy is employed at the convolutional layers so that the signal dimensions are maintained. The activation function is set to ReLU in all hidden layers and softmax at the output layer. Each convolutional layer is followed by a max pooling layer with a kernel size of 2 by 2 and a stride size of 2 by 2. The number of filters in convolution layers is set to 8, 16, and 32, respectively. The number of hidden nodes in fully connected MLP layers is set to 256 and 128, respectively. Dropout regularization is performed at each hidden layer in which the dropout probability set to 0.7. L2-norm regularization (i.e. weight decay) is also employed on the weight parameters with the regularization coefficient set to 0.001. The network is trained using the gradient descent algorithm with mini-batch size set to 128 and number of iterations to 100,001. The train set is shuffled and a random mini-batch is employed in each iteration. Weights are randomly initialized by the Xavier approach [35]. Learning rate is initialized to 0.1 and an exponential decay is performed for the learning rate with decay step set to 1000 and decay rate to 0.96. The loss function is selected as the cross-entropy, which is minimized to learn the weight parameters of the network. Tensorflow is executed on three different settings: single-core CPU, multi-core CPU (number of cores set to 28) and GPU. Once trained, the network was able to obtain 96.3% classification accuracy on test set. Tensorflow version 1.10.0 is employed in AGÜ HPC, 1.5.0 in İTÜ UhEM, and 1.12.0 in all the remaining systems. Similar to scikit-learn, the software version differences do not contribute significantly to running time performance of Tensorflow. For example in TRUBA, version 1.12.0 is used, which is the most recent among the versions tested though the running times of Tensorflow are not the smallest in this system.

2.1.7. Support Vector Machine by libSVM and ThunderSVM

libSVM [14] is developed in C++ programming language for implementing support vector machine (SVM) models [36], which runs on single-core CPU only. Recently, an alternative software named ThunderSVM [15] is introduced that parallelizes the kernel computation steps of an SVM. ThunderSVM is developed in C++ and can be executed on multi-core CPU and on GPU systems.

In this paper, a support vector machine model is implemented using libSVM and ThunderSVM for protein secondary structure prediction. Figure 7 shows the principle behind a support vector machine, which maps the input feature vectors to a new space by a kernel transformation and finds a linear hyper-plane that best separates data samples.

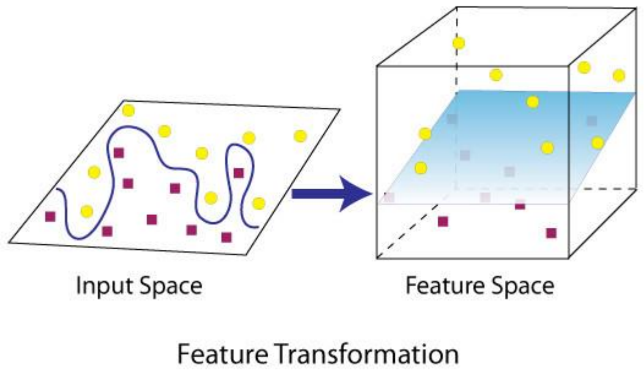


Figure 7. A support vector machine maps data samples to a higher dimensional space and finds a hyper-plane that best separates classes [37]

Given a training set of sample-label pairs $(\mathbf{x}_n, y_n)$ $(1 \leq n \leq N)$ where $\mathbf{x}_n \in \mathbb{R}^{\boldsymbol{D}}$ and $y_n \in \{-1,1\}$, an SVM classifier aims to solve the following optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n \tag{6}$$

subject to

$$y_n(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b) \geq 1 - \xi_n \tag{7}$$

$$\xi_n \geq 0 \tag{8}$$

where C is the penalty parameter of the error term, b is the bias parameter, and $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)\phi(\mathbf{x}_j)$ is denoted as the kernel function. In this paper, an RBF (i.e. Gaussian) kernel is employed.

The dataset used to train and test the SVM classifier in this work contains position specific scoring matrix (PSSM) features obtained by PSI-BLAST and HHblits alignment methods as well as structural profile matrices. The train set contains 36,676 samples and the test set contains 10,497 samples. Each data sample corresponds to an amino acid of a protein. The number of features is 473 and the number of class labels is 3. In all experiments, libSVM version 3.21 is employed.

2.1.8. Dynamic Bayesian Network by GMTK

Graphical Models Toolkit (GMTK) [13] is developed in C++ programming language for implementing probabilistic graphical models by Bilmes lab [38]. In this paper, a dynamic Bayesian network (DBN) [39], which is a time-series lattice model (a super-class of hidden Markov model [40]), is implemented using GMTK for protein secondary structure, solvent accessibility, and torsion angle class prediction problems [41]. A DBN model is shown in Figure 8. Predicting such properties of proteins is widely used as precursors for predicting the three dimensional structure, which enables to elucidate the functional role of the protein and has applications in drug design.

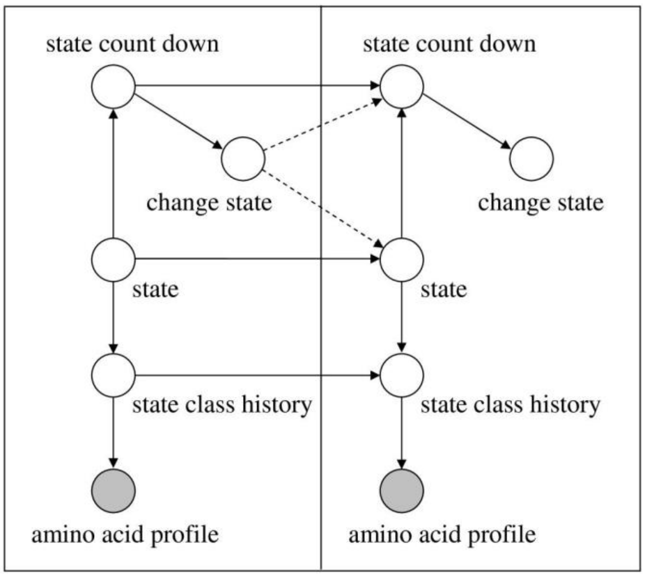


Figure 8. A dynamic Bayesian network is a probabilistic graphical model

For secondary structure prediction, two benchmark datasets named CB513 [42] and EVAset [43] are employed. For solvent accessibility and torsion angle class prediction, the EVAset benchmark is employed. 220 proteins are selected randomly from CB513 to form a train set, and another 220 to form a test set. As a result of this selection, the number of amino acid samples is obtained as 36,946 for train set and 36,676 for test set derived from CB513. The number of input features used to train each conditional Gaussian distribution of the DBN model is 120. A similar selection procedure is applied to EVAset. As a result, 2589 proteins with 532,216 amino acid samples are randomly selected to form the train set and 287 proteins with 52,379 amino acid samples to form the test set. The number of input features that are employed to train conditional Gaussian distributions of the DBN is 200 for solvent accessibility and torsion angle prediction experiments performed on EVAset. Details of the DBN model implemented for predicting structural properties of proteins can be found in the papers by Aydin et al. [44], [45]. All the experiments are performed on single-core CPU using GMTK version 1.4.4.

2.2. Hardware Resources

In this section, we explain the hardware specifications of the computing systems used in this work.

2.2.1 TRUBA

TRUBA (Turkish National Science e-Infrastructure) also known as TÜBİTAK ULAKBIM High Performance and Grid Computing Center [46] is one of the national clusters of Turkey located in the city of Ankara. It contains various hardware resources and SLURM job queues (i.e. partitions). The software simulations performed in this paper are executed in different queues of TRUBA and on machines with different capacities. The jobs that used single-core CPU are executed on the single partition, those that employed multi-core CPUs are executed on the short partition, and those that required GPU are executed on the akya-cuda partition. The jobs that are sent to single partition are executed on machines named levrek, which have the following specifications: 32 CPUs, 2 sockets, 8 cores per socket, 2 threads per core, 2 nodes, CPU model Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz, and 256 GB of RAM. The multi-core CPU versions of PSI-BLAST, HHblits, ThunderSVM, WEKA, and scikit-learn are executed on the short partition on machines named barbun, which have the following configurations: 80 CPUs, 2 sockets, 20 cores per socket, 2 threads per core, CPU model Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, and 384 GBs of RAM. The multi-core CPU versions of Tensorflow are executed on the short partition on machines called sardalya with the following maximum specifications: 56 CPUs, 2 sockets, 14 cores per socket, 2 threads per socket, CPU model Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, and 256 GBs of RAM. The GPU versions of ThunderSVM and Tensorflow are executed on akya-cuda partition which have the following maximum CPU configurations: 40 CPUs, 2 sockets, 20 cores per socket, 1 threads per core, CPU model Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, 384 GBs of RAM. The GPU configuration of akya-cuda includes 4 NVIDIA Tesla V100 with NVlink connection interface and 16 GBs of RAM. Details of hardware specifications in TRUBA can be found on the wiki page [47].

2.2.2 İTÜ UHeM

UHeM [48] is established in Istanbul Technical University (İTÜ). It is known as National Center for High Performance Computing funded by the Ministry of Development. It has a distributed cluster system as shown in Figure 9.

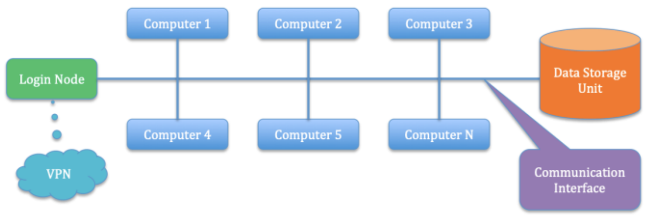


Figure 9. Distributed cluster system of İTÜ UhEM

Similar to TRUBA, UHeM contains various hardware resources and SLURM job queues (i.e. partitions). The SLURM jobs of this paper are executed on shortq, defq, bigmemq and gpuq partitions of UHeM's Sariyer cluster. Each CPU server employed for the present work had the following specifications: 28 CPUs, 2 sockets, 14 cores per socket, 1 threads per core, CPU model Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz. The RAM sizes of these machines are typically 128 GB (except for machines having name range f004-f013 which have 512 GB corresponding to the queue named bigmemq). The gpuq partition contains NVIDIA Tesla K20m GPUs. Detailed hardware specifications of Sariyer cluster can be found on the wiki page [49].

2.2.3 Feynman Grid

Feynman Grid is the High Performance Computing cluster system of CompecTA company in Istanbul, Turkey [50]. All the CPU jobs in this work are executed in the short partition of Feynman Grid and the GPU jobs are executed on the cuda partition. The compute nodes have the following hardware specifications: 56 CPUs, 2 sockets, 14 cores per socket, 2 threads per core, CPU model Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz. The RAM capacity of each compute node is 128 GB. The GPU node contains 4 NVIDIA Tesla K80.

2.2.4 AGÜ HPC

AGÜ HPC is the High Performance Computing cluster established in Abdullah Gul University [51]. All CPU jobs are executed in shorter partition except for the CPU version of Tensorflow which is executed on the short partition. The hardware specifications of the compute nodes are as follows: 36 CPUs, 2 sockets, 18 cores per socket, 1 threads per core, CPU model Intel(R) Xeon(R) Gold 6150 CPU @ 2.70GHz. The RAM size in each compute node is 384 GBs. The GPU jobs are executed in cuda partition, which contains NVIDIA DGX-1, with V100 processor, 8 GPUs, NVlink connection interface and 512 GBs of RAM. Detailed specifications of DGX-1 can be found in [9].

2.2.5 Fujitsu Workstation

The Fujitsu workstation contains 32 CPUs, 2 sockets, 8 cores per socket, 2 threads per core, 2 nodes, CPU model Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz, and 64 GBs of RAM. As the GPU, Fujitsu has one NVIDIA Tesla K20c and one Quadro K2000.

2.2.6 Supermicro Workstation

The Supermicro workstation contains 28 CPUs, 2 sockets, 14 cores per socket, 1 thread per core, 2 nodes, CPU model Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, and 128 GBs of RAM. This workstation also has two NVIDIA Tesla K80 GPUs.

3. FINDINGS

In this section, software execution times on various systems as well as the dependency of selected software on the number of CPUs are analyzed. Since there is no work in the literature that perform a similar analysis on the selected software and system configurations, the performance results obtained for Fujitsu and Supermicro workstations are taken as the baseline while the rest represent the HPC systems.

3.1 Speed Comparison of Systems

This section includes the running times of software on different systems. Tables 1 and 3 shows the running times of two bioionformatics software for protein sequence alignment, each executed twice with the second execution of a given software is performed right after the first execution finished. In these tables, MC stands for multi-core CPU in which 28 threads are used. According to Table 1, the best running times are obtained on Supermicro workstation (1st execution), TRUBA and AGÜ HPC (2nd execution). A large time difference is obtained between the first and the second executions of PSI-BLAST. This is due to the fact that the first time the program is executed, it loads the NR database with hundreds of millions of proteins to RAM, which is the main performance bottleneck. When the program is executed again soon after the first execution, the running time drops considerably due to the availability of the database in RAM's cache unless a script in the system clears the cache automatically. An exception occurred for Fujitsu workstation which had a RAM capacity of 64 GBs where the NR database with size more than 90 GBs did not fit into RAM. For this reason, the second execution took much longer on this workstation as compared to other systems. A similar behavior is observed for UhEM. However the UhEM's PSI-BLAST job is submitted to bigmemq partition which has 512 GBs of RAM capacity. Therefore the reason for having the second execution taking a long time may be attributed to the inavailability of the NR database in RAM's cache. This can be due to the fact that bigmemq is a partition that is designed for jobs that require high RAM capacity and automated system scripts may be preventing the jobs to use the cache in subsequent executions. Table 2 lists the partitions and machines (i.e. compute nodes) used to execute PSI-BLAST on various systems. Note that since Fujitsu and Supermicro are workstation computers, they only contain one SLURM partition (i.e. no multiple partitions or nodes are defined on these machines).

Table 1. Running times of PSI-BLAST on various computing systems. A second submission is performed right after the first submission finished executing.

System	PSI-BLAST 1 MC	PSI-BLAST 2 MC
TRUBA	11 min 45 sec	**1 min 0 sec**
UhEM	30 min 01 sec	30 min 18 sec
Feynman	9 min 30 sec	2 min 22 sec
AGÜ	22 min 26 sec	**1 min 0 sec**
Fujitsu	41 min 38 sec	42 min 22 sec
Supermicro	**4 min 37 sec**	1 min 48 sec

Table 2. Partitions and nodes used for PSI-BLAST on various computing systems. A second submission is performed right after the first submission finished executing.

System	Partition, Node
TRUBA	short, barbun[68-70]
UhEM	bigmemq, f008
Feynman	short, cn03
AGÜ	shorter, cn01

Table 3 summarizes the running times of HHblits. The best results are obtained on Supermicro Workstation (1st execution) and UhEM (2nd execution). Similar to PSI-BLAST, there has been a difference between the running times of the first and the second execution. However this difference is less than PSI-BLAST, which could be related to the fact the protein sequence database employed in HHblits (i.e. Uniprot) is smaller than the NR database of PSI-BLAST. Table 4 lists partitions and machines (i.e. compute nodes) used to execute HHblits on various systems. Note that on UhEM, shortq partition is used instead of bigmemq since the database of HHblits is not as large as the NR database of PSI-BLAST. For this reason, the running time of the second execution has reduced as compared to the PSI-BLAST experiments on UhEM.

Table 3. Running times of HHblits on various computing systems. A second submission is performed right after the first submission finished executing.

System	HHblits 1 MC	HHblits 2 MC
TRUBA	1 min 10 sec	35 sec
UhEM	1 min 06 sec	**29 sec**
Feynman	1 min 18 sec	1 min 03 sec
AGÜ	1 min 19 sec	1 min 0 sec
Fujitsu	1 min 37 sec	56 sec
Supermicro	**45 sec**	44 sec

Table 4. Partitions and nodes used for HHblits on various computing systems. A second submission is performed right after the first submission finished executing.

System	Partition, Node
TRUBA	short, barbun
UhEM	shortq, f001
Feynman	short, cn07
AGÜ	shorter, cn01

Table 5 contains the running time of the random forest model implemented by the scikit-learn library of Python. Reading data from disk to RAM, model training, and prediction times are evaluated separately both on single-core (SC) and multi-core (MC) CPUs, in which 28 threads are used for parallel processing. The best data upload time is obtained on Fujitsu workstation, and the best model training and prediction times are obtained on AGÜ HPC. Table 6 lists partitions and compute nodes used to run scikit-learn on various systems.

Table 5. Running times of random forest model of scikit-learn library of Python on various computing systems. Data upload, model training, and prediction are evaluated on single and multiple core CPUs.

System	Load SC	Train SC	Predict SC	Train MC	Predict MC
TRUBA	1 min 24 sec	5 min 28 sec	14 sec	3 min 42 sec	11 sec
UhEM	1 min 22 sec	3 min 55 sec	10 sec	35 sec	12 sec
Feynman	1 min 31 sec	6 min 36 sec	25 sec	40 sec	11 sec
AGÜ	56 sec	**3 min 46 sec**	**7 sec**	**17 sec**	**4 sec**
Fujitsu	**38 sec**	5 min 4 sec	12 sec	38 sec	14 sec
Supermicro	42 sec	3 min 57 sec	9 sec	32 sec	11 sec

Table 6. Partitions and nodes used for scikit-learn on various computing systems.

System	Partition, Node
TRUBA	single, levrek4
UhEM	shortq, s052
Feynman	short, cn01
AGÜ	shorter, cn01

Table 7 displays the running times of the multi-layer perceptron (MLP) neural network model implemented by the WEKA software. The time in each cell includes model training and testing since these operations are performed by a single line of command in WEKA. The best single-core running time is obtained on UhEM and the best multi-core running time on AGÜ HPC. The partitions and compute nodes used in this experiment are summarized on Table 8.

Table 7. Running times of multi-layer perceptron model of WEKA on various computing systems. Model is trained and tested on the same dataset. Combined model train and prediction times are evaluated on single and multiple core CPUs.

System	MLP SC	MLP MC
TRUBA	2 min 41 sec	5 min 27 sec
UhEM	**2 min 9 sec**	6 min 19 sec
Feynman	2 min 46 sec	6 min 44 sec
AGÜ	2 min 53 sec	**2 min 54 sec**
Fujitsu	2 min 32 sec	6 min 6 sec
Supermicro	6 min 33 sec	-----

Table 8. Partitions and nodes used for WEKA on various computing systems.

System	Partition, Node
TRUBA	single, levrek4
UhEM	shortq, s052
Feynman	short, cn01
AGÜ	shorter, cn01

Table 9 includes the running times of the deep convolutional neural network model implemented by the Tensorflow library of Python on single-core CPU (SC), multi-core CPU (MC), and GPU. The multi-core CPU experiments are performed using 28 threads. Model training and prediction times are evaluated together and the running times are mostly dominated by model training since prediction takes much shorter than training.

The best running times are obtained on Supermicro workstation for single-core and multi-core CPU and on AGÜ HPC for GPU. The GPU system of TRUBA gave segmentation fault error and that of Feynman Grid was on maintenance at the time these experiments were performed. The single core execution on Feynman Grid also gave error and could not be evaluated (the job was killed with no reason). The NVIDIA's DGX-1 system performed three times faster than the NVIDIA's Tesla K20c and K80 models available on the workstations. Table 10 contains the partitions and nodes used in Tensorflow experiments.

Table 9. Running time of deep convolutional neural network model of Tensorflow library of Python on various computing systems. The running times include model training and prediction on single-core, multiple core CPUs and GPU.

System	CNN SC	CNN MC	CNN GPU
TRUBA	4h 48 min 55 sec	1 h 13 min 36 sec	-----
UhEM	5 h 03 min 12 sec	1 h 11 min 05 sec	19 min 2 sec
Feynman	-----	3 h 27 min 05 sec	-----
AGÜ	5 h 35 min 29 sec	3 h 24 min 06 sec	**4 min 39 sec**
Fujitsu	2 h 47 min 18 sec	2 h 10 min 09 sec	13 min 57 sec
Supermicro	**1 h 34 min** 33 sec	**37 min 04 sec**	12 min 32 sec

Table 10. Partitions and nodes used for Tensorflow on various computing systems.

System	Partition, Node (CPU)	Partition, Node, (GPU)
TRUBA	single, levrek110	akya-cuda, akya19
UhEM	defq, s001	gpuq, f001
Feynman	long, cn01	-----
AGÜ	short, cn07	cuda, dgx01

Table 11 shows the running times of support vector machine model implemented by libSVM, which only operates on single-core CPU. Model training and prediction steps are evaluated separately since these require separate lines of commands. The best running times are obtained by Supermicro workstation though other systems also gave similar performance. The SLURM partitions and compute nodes used in these experiments are summarized in Table 12.

Table 11. Running times of support vector machine model of libSVM on various computing systems. The running times for model training and prediction are obtained separately on single-core CPU.

System	SVM Train	SVM Predict
TRUBA	10 min 10 sec	2 min 27 sec
UhEM	9 min 31 sec	2 min 14 sec
Feynman	16 min 04 sec	3 min 03 sec
AGÜ	10 min 56 sec	2 min 23 sec
Fujitsu	10 min 29 sec	2 min 32 sec
Supermicro	**9 min 07 sec**	**2 min 05 sec**

Table 12. Partitions and nodes used for libSVM on various computing systems.

System	Partition, Node
TRUBA	single, levrek112
UhEM	defq, s076
Feynman	short, cn03
AGÜ	shorter, cn01

Table 13 presents the running times of the support vector machine model implemented by ThunderSVM on single-core CPU (SC), multi-core CPU (MC) and GPU. Model training and prediction times are evaluated separately. The best results for single-core model training, single-core prediction and multi-core CPU model training are obtained on UhEM and the best results for multi-core CPU model prediction as well as GPU based model training and prediction are obtained on TRUBA.

The GPU running time of DGX-1 is also obtained as close to TRUBA's recently established GPU system, which also had V100 processors and NVlink communication interface. Since the SVM job did not require significant amount of RAM the GPU servers performed comparably well. SLURM partitions and compute nodes used in these experiments are listed in Table 14.

Table 13. Running times of support vector machine model of ThunderSVM on various computing systems. Model training and prediction times are evaluated on single core CPU, multi core CPU and GPU.

System	Train SC	Pred SC	Train MC	Pred MC	Train GPU	Pred GPU
TRUBA	393 sec	57 sec	64 sec	**8 sec**	**9 sec**	**5 sec**
UhEM	**58 sec**	**11 sec**	**55 sec**	11 sec	23 sec	8 sec
Feynman	104 sec	29 sec	159 sec	27 sec	-----	-----
AGÜ	778 sec	113 sec	170 sec	58 sec	13 sec	7 sec
Fujitsu	213 sec	13 sec	209 sec	66 sec	30 sec	10 sec
Supermicro	110 sec	16 sec	109 sec	15 sec	16 sec	6 sec

Table 14. Partitions and nodes used for ThunderSVM on various computing systems.

System	Partition, Node (SC)	Partition, Node (MC)	Partition, Node (GPU)
TRUBA	single, levrek4	single, barbun	akya-cuda, akya9
UhEM	shortq, s025	shortq, s025	gpuq, f003
Feynman	short, cn01	short, cn01-02	-----
AGÜ	shorter, cn01	shorter, cn01	cuda, dgx01

Table 15 includes the running times of dynamic Bayesian network model implemented using GMTK software for secondary structure prediction on CB513 benchmark. Similarly, Tables 17, 19, and 21 include the running times of GMTK on EVAset benchmark for secondary structure prediction, for solvent accessibility prediction, and for torsion angle class prediction, respectively. In all experiments, model training and prediction steps are executed separately on single-core CPU. The best running times are typically obtained on the Supermicro workstation, except for torsion angle class prediction on EVAset, which had the best running times on UhEM. The SLURM queues and compute nodes employed in these experiments are tabulated in Tables 16, 18, 20, and 22.

Table 15. Running times of dynamic Bayesian network model implemented on CB513 benchmark using GMTK for protein secondary structure predicton on various computing systems. The running times for model training and prediction are obtained separately on single-core CPU.

System	DBN Train	DBN Predict
TRUBA	54 sec	14 min 41 sec
UhEM	32 sec	14 min 31 sec
Feynman	39 sec	17 min 07 sec
AGÜ	38 sec	15 min 54 sec
Fujitsu	30 sec	13 min 56 sec
Supermicro	**25 sec**	**12 min 15 sec**

Table 16. Partitions and nodes used for secondary structure prediction on CB513 benchmark using GMTK on various computing systems.

System	Partition, Node
TRUBA	single, levrek7
UhEM	shortq, s070
Feynman	short, cn03
AGÜ	shorter, cn01

Table 17. Running times of dynamic Bayesian network model implemented on EVAset benchmark using GMTK for protein secondary structure predicton on various computing systems. The running times for model training and prediction are obtained separately on single-core CPU.

System	DBN Train	DBN Predict
TRUBA	28 min 05 sec	16 min 02 sec
UhEM	15 min 06 sec	14 min 04 sec
Feynman	18 min 36 sec	18 min 06 sec
AGÜ	16 min 29 sec	17 min 42 sec
Fujitsu	16 min 51 sec	14 min 43 sec
Supermicro	**14 min 10 sec**	**13 min 20 sec**

Table 18. Partitions and nodes used for secondary structure prediction on EVAset benchmark using GMTK on various computing systems.

System	Partition, Node
TRUBA	single, levrek122
UhEM	shortq, f001
Feynman	short, cn01
AGÜ	shorter, cn01

Table 19. Running times of dynamic Bayesian network model implemented on EVAset benchmark using GMTK for protein solvent accessibility prediction on various computing systems. The running times for model training and prediction are obtained separately on single-core CPU.

System	DBN Train	DBN Predict
TRUBA	23 min 46 sec	2 min 52 sec
UhEM	15 min 45 sec	2 min 48 sec
Feynman	19 min 51 sec	3 min 41 sec
AGÜ	18 min 19 sec	3 min 28 sec
Fujitsu	17 min 03 sec	2 min 52 sec
Supermicro	**14 min 53 sec**	**2 min 35 sec**

Table 20. Partitions and nodes used for solvent accessibility prediction on EVAset benchmark using GMTK on various computing systems.

System	Partition, Node
TRUBA	single, levrek122
UhEM	shortq, f005
Feynman	short, cn01
AGÜ	shorter, cn01

Table 21. Running times of dynamic Bayesian network model implemented using GMTK for protein torsion angle class prediction on various computing systems. The running times for model training and prediction are obtained separately on single-core CPU.

System	DBN Train	DBN Predict
TRUBA	17 min 40 sec	1 h 15 min 17 sec
UhEM	**9 min 12 sec**	1 h 15 min 25 sec
Feynman	13 min 08 sec	1 h 27 min 07 sec
AGÜ	12 min 01 sec	1 h 22 min 33 sec
Fujitsu	10 min 59 sec	1 h 12 min 32 sec
Supermicro	9 min 51 sec	**1 h 02 min 40 sec**

Table 22. Partitions and nodes used for torsion angle class prediction on EVAset benchmark using GMTK on various computing systems.

System	Partition, Node
TRUBA	single, levrek122
UhEM	defq, s047
Feynman	short, cn01
AGÜ	shorter, cn01

3.2 Optimizing the number of CPU threads

In this section, three methods are selected that have multi-core CPU implementation available: PSI-BLAST, HHblits and WEKA's MLP method. The software running times are obtained with respect to the number of CPU threads including single core and multi-core CPU options (Figures 10-12). According to Figure 10, there is a consireable increase in performance of PSI-BLAST if the program is executed on multiple CPU cores in parallel as compared to single-core execution, which took 1311 seconds. The best running time is obtained as 106 seconds using 60 threads, a 12 fold improvement in performance as compared to single core execution. A similar behavior is obtained for the HHblits software for which the best running time is obtained as 22 seconds when the number of CPU threads is 48. This is an 8 fold increase in performance as compared to single core execution, which took 183 seconds (Figure 11). On the other hand, the multi layer perceptron model implemented by WEKA software did not benefit much from increasing the number of CPU cores (Figure 12). Almost all cases obtained a similar running time around 5 minutes and 30 seconds with the best running time obtained using 16 CPU threads. This experiment is also repeated using Tensorflow software (both in multi core CPU and GPU) and a similar behaviour is obtained and GPU performance was even worse than CPU (results not shown). Based on these, it can be anticipated that the MLP model implementation in these software may not be quite suitable for parallelization across multiple CPU cores.

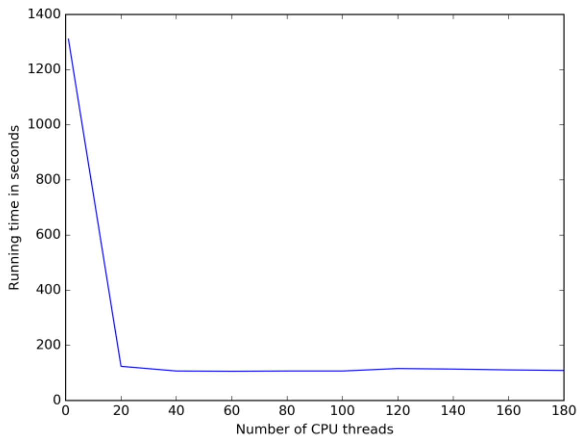


Figure 10. Running time of PSI-BLAST with respect to the number of CPU threads

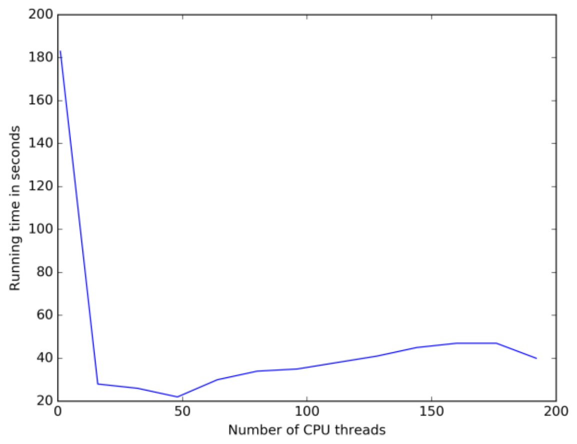


Figure 11. Running time of HHblits with respect to the number of CPU threads

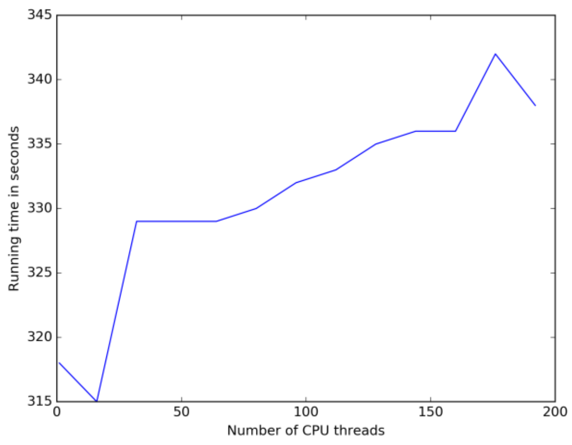


Figure 12. Running time of MLP model implemented using WEKA with respect to the number of CPU threads

4. DISCUSSION

This work provides a comparative and comprehensive speed analysis of selected machine learning and bioinformatics software on various high performance computing systems. The following can be deduced from the analysis provided in this work:

- For single core jobs, workstations with sufficient CPU and RAM resources can perform comparably or better than HPC cluster systems, which are typically loaded with many jobs running simultaneously. This will also reduce the waiting times in the queue of a large system. Therefore to optimize the computational needs of a research lab, both having access to workstation computers and larger cluster systems will be the best choice.
- HPC clusters can be particularly more useful than workstations for programs that can run in multiple CPU or GPU cores or for programs that should be run repeatedly for different parameter settings.
- GPU versions of the programs typically perform better than CPU versions especially if the algorithm is suitable for parallelization across multiple cores. Examples include deep convolutional neural networks and support vector machines. An example exception to this behavior is the MLP neural network model for which the GPU performance is worse than CPU due to training algorithm being less suitable for parallelization.
- In certain cases, CPU nodes can still be preferred over GPU nodes. One example can be hyper-parameter optimization of machine learning models, which requires executing the same algorithm many times each with a different hyper-parameter setting. From a practical stand point, such an optimization can be parallelized across multiple CPU cores more easily (serial parallelization) as the number of GPU nodes in a system will typically be less than the number of CPU nodes and parallelizing across thousands of GPU cores will require more advanced programming skills such as re-implementing the hyper-parameter optimization scripts using more advanced software (e.g. CUDA programming).
- Using GPU systems specialized for the problem of interest can provide significant performance gains. For example, NVIDIA’s DGX-1 developed for artificial intelligence and machine learning applications contains high number of GPU cores, high speed processors, high RAM capacity, and fast communication interface called NVlink enabling faster model training as compared to older GPU models such as Tesla K80.
- As the number of CPU threads are increased, the performance of an application can also increase but may saturate and start to decrease after some point. This could be due to the memory system’s not being able to service data requests efficiently because the processes share the limited resources of cache capacity and memory bandwidth. Scaling can be harmed by memory loading/storing operations. Memory intensive programs can therefore suffer from memory bandwidth saturation. Other factors can include increased I/O requests of increased number of processes and whether the algorithm is suitable for parallelization. Fort his reason, to get the most out of parallel processing, hardware and software conditions should be optimized together.

5. CONCLUSION

This paper presents a comprehensive analysis for the running time performance of popular software on selected research problems and HPC systems. The As a future work, a similar analysis can be performed using other software on different fields and problems such as finance, forecasting; on platforms such as cloud computing; and on larger datasets where RAM size also becomes a bottleneck. Efforts in system performance analysis will provide richer information and guidance to end users for optimizing the performance of their applications.

6. ACKNOWLEDGEMENTS

This work was supported by 3501 TUBITAK National Young Researchers Career Award [grant number 113E550]. The numerical calculations reported in this paper were fully/partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources), the National Center for High Performance Computing of Turkey (UHeM) under project no 5004062016, AGÜ HPC, and Feynman Grid. Special thanks to Mr. Aydın Şaşmaz, the manager of the CompecTA company, for making the Feynman Grid and its resources available. Mr. Kenan Pelit and Mr. Ekrem Seren from CompecTA provided great support as solution architects in installing some of the software tested in this work.

REFERENCES

[1]. R. Bekkerman, M. Bilenko, and J. Langford, Scaling Up Machine Learning: Parallel and Distributed Approaches, Cambridge University Press, 2012.
[2]. Supercomputer, https://en.wikipedia.org/wiki/Supercomputer (first published on 31 January 2002).
[3]. Y. Kochura, S. Stirenko, O. Alienin, M. Novotarskiy, and Y. Gordienko, “Performance Analysis of Open Source Machine Learning Frameworks for Various Parameters in Single-Threaded and Multi-Threaded Modes”, In: Shakhovska N., Stepashko V. (eds) Advances in Intelligent Systems and Computing II. CSIT 2017. Advances in Intelligent Systems and Computing, vol 689. Springer, 243-256, 2018. DOI: https://doi.org/10.1007/978-3-319-70581-1_17.
[4]. V. Kovalev, A. Kalinovsky, and S. Kovalev, “Deep Learning with Theano, Torch, Caffe, TensorFlow, and Deeplearning4J: Which One Is the Best in Speed and Accuracy?”, International Conference on Pattern Recognition and Information Processing, (2016). http://elib.bsu.by/handle/123456789/158561.
[5]. A. Shatnawi, G. Al-Bdour, R. Al-Qurran, and M. Al-Ayyoub, “A Comparative Study of Open Source Deep Learning Frameworks”, IEEE 9th International Conference on Information and Communication Systems (ICICS), 72-77, (2018). DOI: 10.1109/IACS.2018.8355444.
[6]. S. Bahrampur, N. Ramakrishnan, L. Schott, and M. Shah, “Comparative Study of Deep Learning Software Frameworks”, arXiv:1511.06435, 2016.
[7]. D.A. Bader, Y. Li, T. Li, and V. Sachdeva, “BioPerf: A Benchmark Suite to Evaluate High-Performance Computer Architecture on Bioinformatics Applications”, The IEEE International Symposium on Workload Characterization (IISWC 2005), Austin, TX, October 6-8, 2005. DOI: 10.1109/IISWC.2005.1526013.
[8]. M. Kurtz, F. J. Esteban, P. Hernandez, J. A. Caballero, A. Guevara, G. Dorado, and S. Galvez, “Bioinformatics Performance Comparison of Many-core Tile64 vs. Multi-core Intel Xeon”, Clei Electronic Journal, vol. 17, no. 1, 1-9, 2014.
[9]. NVIDIA DGX-1, https://www.nvidia.com/en-us/data-center/dgx-1/ (published on 9 October 2017).
[10]. M. Abadi et al., “Tensorflow: A system for large-scale machine learning”, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)”, USENIX Association, 265-283, (2016). Software available at https://www.tensorflow.org. (published on 5 March 2019).
[11]. F. Pedregosa et al., “Scikit-learn: machine learning in python”, Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011. Software available at https://scikit-learn.org/stable/ (published on 20 October 2011). Url: http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.
[12]. E. Frank, M. A. Hall, and I. Witten, “The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016. Software available at https://www.cs.waikato.ac.nz/ml/weka/ (published on 13 July 2008).
[13]. J. Bilmes and G. Zweig, “The graphical models toolkit: An open source software system for speech and time-series processing”, IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 4, IV-3916-IV-3919, (2002). Software available at https://melodi.ee.washington.edu/gmtk/ (published on 20 October 2014). DOI: 10.1109/ICASSP.2002.5745513.
[14]. C.-C. Chang and C.-J. Lin, “LIBSVM: a library for support vector machines”, ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 27:1--27:27, 2011. Software available at https://www.csie.ntu.edu.tw/~cjlin/libsvm/ (published on 15 July 2018). DOI: 10.1145/1961189.1961199.
[15]. Z. Wen, J. Shi, Q Li, B. He, and J. Chen, “ThunderSVM: A Fast SVM Library on GPUs and CPUs”, Journal of Machine Learning Research, vol. 19, pp. 1-5, 2018. Software available at https://thundersvm.readthedocs.io/en/latest/ (published on 2 November 2017).
[16]. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", Nucleic Acids Res. 25 (17), 3389-3402, (1997). Software available at https://blast.ncbi.nlm.nih.gov/Blast.cgi (published on 30 May 2019). DOI: 10.1093/nar/25.17.3389.
[17]. M. Remmert, A. Biegert, A. Hauser, and J. Söding, "HHblits: Lightning-fast iterative protein sequence searching by HMM-HMM alignment", Nat. Methods, 9 (2), 173-175, (2011). Software available at https://github.com/soedinglab/hh-suite (published on 27 February 2019). DOI: 10.1038/nmeth.1818.
[18]. NCBI, URL: https://www.ncbi.nlm.nih.gov (first published on Nov. 4, 1988).
[19]. Protein Data Bank (PDB), https://www.rcsb.org (published on 23 April 2019).
[20]. D. T. Jones, “Protein secondary structure prediction based on position-specific scoring matrices”, Journal of Molecular Biology, vol 292, no. 2, 195-202, 1999. Software available at http://bioinf.cs.ucl.ac.uk/psipred/ (published on 26 June 2009). DOI: 10.1006/jmbi.1999.3091.
[21]. DSSP, URL: https://swift.cmbi.umcn.nl/gv/dssp/DSSP_1.html, (first published in 1983).
[22]. Python, https://www.python.org (published on 22 February 2014).
[23]. F. Bulut, “Sınıflandırıcı Topluluklarının Dengesiz Veri Kümeleri Üzerindeki Performans Analizleri”, Bilişim Teknolojileri Dergisi, 9(2), 153, 2016. DOI: 10.17671/btd.81137.
[24]. Artnome, https://www.artnome.com/news/2018/11/8/inventing-the-future-of-art-analytics (published on 12 November 2018).
[25]. F. Bulut, “Çok katmanlı algılayıcılar ile doğru meslek tercihi”, Anadolu Üniversitesi Bilim Ve Teknoloji

Dergisi A-Uygulamalı Bilimler ve Mühendislik, 17(1), 97-109, 2016. DOI: 10.18038/btda.45787.
[26]. Multi-layer perceptron, https://www.oreilly.com/library/view/getting-started-with/9781786468574/ch04s04.html (published on 4 August 2016).
[27]. S. Fourati et al., "A crowdsourced analysis to identify ab initio molecular signatures predictive of susceptibility to viral infection", Nature Communications, vol. 9, no. 1, pp. 1-11, 2018. Challenge web site: https://www.synapse.org/#!Synapse:syn5647810/wiki/399103 (first published on 17 May 2016). DOI: 10.1038/s41467-018-06735-8.
[28]. C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1996.
[29]. Convolutional neural network, https://en.wikipedia.org/wiki/Convolutional_neural_network (first published on 31 August 2013).
[30]. Optical character recognition, https://en.wikipedia.org/wiki/Optical_character_recognition (first published on 7 December 2005).
[31]. A comprehensive guide to convolutional neural networks, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (published on 15 December 2018).
[32]. notMNIST dataset, http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html (published on 8 September 2011).
[33]. MNIST dataset, https://en.wikipedia.org/wiki/MNIST_database (first publised on 17 August 2013).
[34]. Using notMNIST dataset from Tensorflow, http://enakai00.hatenablog.com/entry/2016/08/02/102917 (published on 2 August 2016).
[35]. X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), 249-256, (2009). Available at http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.
[36]. Support vector machine, https://en.wikipedia.org/wiki/Support-vector_machine (first published on 27 July 2002).
[37]. W. Yu, T. Liu, R. Valdez, M. Gwinn, and M. J. Khoury, "Application of support vector machine modeling for prediction of common diseases: the case of diabetes and pre-diabetes", BMC Medical Informatics and Decision Making, vol. 10, no. 1, 2010. DOI: 10.1186/1472-6947-10-16.
[38]. J. A. Bilmes, http://melodi.ee.washington.edu/~bilmes/pgs/index.html (published on 04 June 2018).
[39]. Dynamic Bayesian network, https://en.wikipedia.org/wiki/Dynamic_Bayesian_network (first publised on 4 December 2004).
[40]. Hidden Markov model, https://en.wikipedia.org/wiki/Hidden_Markov_model (first published on 3 October 2002).
[41]. Protein structure prediction, https://en.wikipedia.org/wiki/Protein_structure_prediction (published on 21 March 2007).
[42]. J. A. Cuff and G. J. Barton, "Evaluation and improvement of multiple sequence methods for protein secondary structure prediction", Proteins, 34(4), 508–519, 1999. Dataset is available at http://www.compbio.dundee.ac.uk/jpred/legacy/data/ (first published in 1999).
[43]. I. Y. Y. Koh, V. A. Eyrich, M. A. Marti-Renom, D. Przybylski, M. S. Madhusudhan, N. Eswar, O. Graña, F. Pazos, A. Valencia, A., and B. Rost, "EVA: Evaluation of protein structure prediction servers", Nucleic Acids Research, 31(13), 3311–3315, 2003. DOI: 10.1093/nar/gkg619.
[44]. Z. Aydin, A. Singh, J. Bilmes and W. S. Noble, "Learning sparse models for a dynamic Bayesian network classifier of protein secondary structure," BMC Bioinformatics, 12:154, 2011. DOI: https://doi.org/10.1186/1471-2105-12-154.
[45]. Z. Aydin, N. Azgınoglu, H. I. Bilgin, and M. Celik, "Developing Structural Profile Matrices for Protein Secondary Structure and Solvent Accessibility Prediction", accepted to Bioinformatics, 2019. DOI: 10.1093/bioinformatics/btz238.
[46]. TRUBA, https://www.truba.gov.tr/index.php/en/main-page/ (first published in 2003).
[47]. TRUBA wiki page, http://wiki.truba.gov.tr/index.php/Ana_sayfa (first published on 1 December 2013).
[48]. UhEM, http://www.uhem.itu.edu.tr (published in 2016).
[49]. İTÜ UhEM wiki page, http://wiki.uhem.itu.edu.tr/w/index.php/Sarıyer_sistemine_iş_vermek (first published on 19 October 2016).
[50]. CompecTA, https://www.compecta.com.tr (first published in 2007).
[51]. Abdullah Gul University, http://www.agu.edu.tr (first published in 2015).