K-NN algorithm in Parallel VLSI Simulation Scheduling

Qing XU
School of Computer Science
McGill University
Montreal, Quebec H3A 2A7
Email: qxu2@cs.mcgill.ca

Carl Tropper
School of Computer Science
McGill University
Montreal, Quebec H3A 2A7
Email: carl@cs.mcgill.ca

Abstract— **Parallel discrete event simulation has been established as a technique which has great potential to speed up the execution of gate level circuit simulation. A fundamental problem posed by a parallel environment is the decision of whether it is best to simulate a particular circuit sequentially or on a parallel platform.Furthermore, in the event that a circuit should be simulated on a parallel platform, it is necessary to decide how many computing nodes should be used on the given platform. In this paper we propose a machine learning algorithm as an aid in making these decisions. The algorithm is based on the well-known K-Nearest Neighbor algorithm. After an extensive training regime, it was shown to make a correct prediction 99% of the time on whether to use a parallel or sequential simulator. The predicted number of nodes to use on a parallel platform was shown to produce an execution time which within 11% of the smallest execution time. The configuration which resulted in the minimal execution time was picked 61% of the time.**

1 INTRODUCTION

The problems of deciding whether a given discrete event simulation will benefit from a parallel execution and of deciding how many nodes of a given parallel platform to use in its execution is a problem of basic importance in parallel simulation. It also has been paid scant (if any) attention to by the parallel simulation community. In this paper, we present an algorithm which may be used for both of these problems in the domain of parallel gate level circuit simulation.

It has been established that parallel discrete event simulation has the ability to speed up the execution of gate level simulations [12] [13] [14] [4]. Due to different characteristics of circuits, some circuits can benefit from parallel simulation, while other circuits are best simulated sequentially. If parallel simulation is appropriate for a circuit, it is important to know how many nodes of the parallel platform on which they are to be simulated should be used. It is common for organizations devoted to computer aided design (CAD) of VLSI circuitry to make use of a cluster of computers on which to execute simulations. In this environment, simulation tasks are submitted to a central scheduler and are queued by the scheduler prior to their execution.

The algorithm presented in this paper makes use of machine learning, and is illustrated in (see Fig 1). A machine learning(ML) engine is built to use the characteristics of circuits as input and to generate the decision on which software to use(parallel or sequential simulator) and what hardware to use(how many nodes should be used).

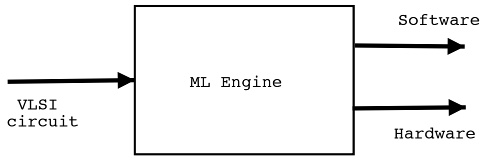

Fig. 1. Generating Parallel Distributed Simulation Scheduling Options

While we confine ourselves to optimistic simulation in this paper, the methodology which we describe can be used for any flavor of parallel simulation. We make use of XTW [12] as our parallel simulator. Our choice of hardware is between a single Sun server and a cluster of workstations. The remainder of this paper is organized as follows. Section 2 describes the K-Nearest Neighbor algorithm and its use in pattern classification problems. Section 3 describes our adaptation of this algorithm to gate level simulation Section 4 describes the experiments which we conducted. Finally, our conclusion is presented in the final section 5.

2 THE K-NEAREST NEIGHBOR ALGORITHM

The K-NN algorithm is an instance based machine learning algorithm. It is used to classify objects according to various features which the objects possess and makes use of training samples to do do the actual classification. The objects are are represented in a so-called feature space as N dimensional vectors whose entries consist of numerical values for the features chosen to characterize the objects (hence the name feature space). The dimension of the feature space is equal to the number of features which are selected to represent an object. The K-NN algorithm classifies objects as belonging to a particular category by simply determining the majority category among an objects' K nearest neighbors. The metric for determining the distance between objects is often the Euclidean distance from the between the objects, although the manhattan distance is also a popular metric. In the event that K=1, an object is assigned to the class of its nearest neighbor, and the algorithm is referred to as the nearest neighbor algorithm. Fig 2 contains an illustration of the algorithm. The query point is in the center of the two circles. If, for example, K=3 then the query object is classified by the 3 objects in the inner circle, while if K=5, it is classified by the objects in the first two circles.

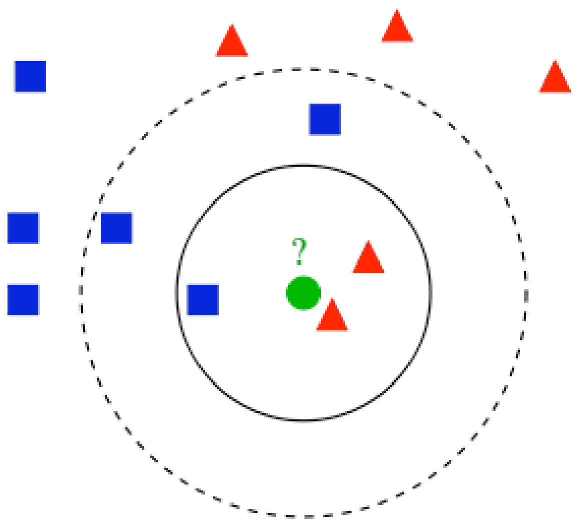

Fig. 2. Classify a Query Point using K-Nearest Neighbor

In order to classify and object it is necessary to start with a collection of objects which are already classified. Creating this set of objects is the training phase of the algorithm. It consists of computing and storing the feature vectors of the collection of objects which are chosen for training the algorithm as well as the classes to which these vectors belong. This phase of the algorithm is then followed by the classification phase, in which objects are classified as belonging to the majority class corresponding to a given distance.

An obvious drawback of the algorithm is the greedy nature of the majority function used to classify objects. It is necessary to provide a good training set to avoid classifying too many objects in the same category. Another approach is to take the distances of the training samples into account when classifying objects. The quality of the algorithm is also dependent on the choice and the number of features selected as well as the method employed to scale the features. For example, evolutionary algorithms are often employed for scaling the features along with the use of mutual information, a technique which is used to determine the correlation of the features.

3 The K-NN Algorithm for PDES Scheduling

In this section we apply the K-NN algorithm to gate level circuit simulation. Circuits are typically designed using a high level language such as Verilog or VHDL. After creating a register transfer level design, the circuit is synthesized to produce a gate level netlist. Our algorithm makes use of this netlist. The objective of the algorithm is to determine whether a given circuit should be simulated serially or in parallel and to determine the number of nodes of the parallel platform to employ if it is to be simulated in parallel. We begin by describing the attributes of a circuit which we have chosen to employ as the features in our feature vector.

3.1 Circuit Attributes

It is certainly possible to chose many attributes in order to define a feature vector. A partial list might include the number of combinational gates, the number of registers, the shortest path through circuit, , number of modules, the average depth of hierarchy tree for a module, i.e. how many modules are included in a bigger module, and the amount of analog circuitry. The list can be extended to an arbitrary length. However, it is well known that the accuracy of the K-NN algorithm can be severely compromised by the presence of irrelevant (or for that matter, noisy) data. It can also be severely compromised if the features do not scale appropriately. Finally, the computational load of the algorithm is greatly increased by each new feature. With these limitations in mind we chose the following attributes to describe our feature vector:

- The number of gates in a circuit. This attribute reflects the size of a circuit
- The number of module instances in a circuit. Circuits are normally designed in a top-down, hierarchical manner. Each functional block of the circuit (e.g. ALU, RAM) is described by a module in a high level language such as Verilog. The same module can be repeated a number of times throughout the design. Each such repetition is termed an instance. Hence the number of module instances is a rough indication of how complex a circuit is. It is clear that the interconnection of the modules which describe a circuit would also be an indicator of the complexity of a design, but it is difficult to find a one dimensional representation for the complexity of the interconnection.
- The number of strings. In this paper, we make use of a string partitioning algorithm in order to partition the circuit and assign the partitions to processors. The string partitioning algorithm starts with primary input gates and descends through the circuit tree in a depth-first search(DFS) manner. When the DFS hits a primary output or a gate which has already been reached, it stops making its way through the circuit and starts a new search. The gates collected in one DFS search are grouped in a string. The number of strings is defined as the number of times the DFS was conducted in partitioning the circuit. This attribute is intended to indicate how much parallelism we can exploit in a circuit for the parallel simulation of the circuit. It should be noted that there are many partitioning algorithms available [8] [9] [2] [1] [3] The objective in partitioning a circuit is to try to find a balance between the load imposed on a node, the communication load imposed on the parallel simulation, and the amount of inherent parallelism in the circuit which can be exploited in the simulation. The upshot of all of this is that if different partitioning algorithms are used, then it may well be necessary to define this feature differently. String partitioning is known for being able to expose the parallelism in a circuit.

3.2 The Feature Vector

As previously mentioned, the first part of the K-NN algorithm consists of training. This training consists of simulating every circuit in our training set on each possible hardware configuration. For each circuit in our training set, the hardware option that generated the smallest simulation time was used to label a training data point. A training data point was labeled with following features:

- Circuit ID
- The number of gates
- The number of module instances
- The number of partition strings
- Software option. This attribute indicate if a parallel simulator or a sequential simulator was used to generate this data point.
- Hardware option. This attribute indicate which computer or computer cluster was used to generate this data point.
- Number of nodes. This attribute indicate how many computer nodes were used to generate this data point. For a sequential run this value is 1.

When we wish to classify a query (i.e. non-training) circuit, we calculate the Euclidean distance to all of the training points. Note, however that the distance is calculated only using the following features- the number of gates, the number of partition strings and the number of module instances. K is set to 1 in our experiments, so the nearest training point is then selected in order to classify the query circuit. It is labeled with the same software option, hardware option and number of nodes as the nearest training point. This classification is employed o simulate the circuit.

4 The Experiments

The hardware which we we made use of in the course of our experiments was a Sun server and a cluster of 20 Linux machines. The Sun server was the Ultra-80 with 4x450MHz Sparcv9 processors with 2G of memory. Each Linux machine in the cluster is equipped with 2.4G-Hz P4 CPU and 1G of memory. The Linux cluster is connected by a 1Gbyte Ethernet switch.

We made use of 60 benchmark circuits in our experiments. Of these circuits, 22 came from the ISCAS89 benchmark suite.The other 38 circuits were artificially generated using a method proposed by Hutton et al [5]. The largest circuit has 360K gates. The smallest circuit has 180 gates. The number of module instances in the circuits varies from 1 to 24. The number of partition strings in the circuits varies from 48 to 70K. Fig 3 contains a portrayal of the characteristics of these circuits.

An optimistic gate level circuit simulator,XTW [11], was used for our parallel simulations. XTW is an outgrowth of Clustered Time Warp [4]. Two of its distinguishing features are that (1)it makes use of a multi-level queueing mechanism and (2) it uses one anti message for message cancellation as opposed to, for example, aggressive message cancellation. It also has a low cost GVT algorithm. XTW combines the following techniques into a framework for parallel simulation.

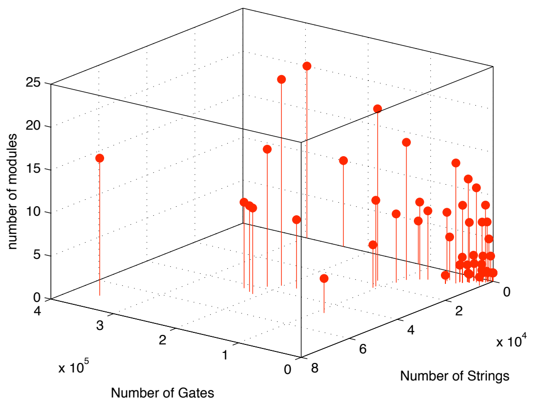

Fig. 3. Benchmark Circuits Feature Value Distributions

- Time Warp – a distributed optimistic synchronization algorithm [6]
- Bounded Time Window – a mechanism that stabilizes Time Warp during run time [15]
- Two color GVT – a low cost Global Snapshot mechanism [10]
- Rollback Relaxation – a way to reduce state saving cost [16]
- Skip List Event Queue – fast event scheduling algorithm
- Event Look Ahead – a mechanism to reduce the number of output events [7]
- Clustered Time Warp – a hybrid distributed simulation system [4]

A sequential version of XTW, SXTW, plays the role of our sequential simulator. SXTW was derived from XTW by removing all of the algorithms in XTW which were related to parallel simulation. These algorithms include state saving, GVT computation, rollback handling, anti-message handling, etc. The event scheduling and the event handling routines are the same as in XTW.

The 60 benchmark circuits were simulated sequentially on the Sun server by SXTW and by XTW on the Linux cluster using from 2 to 20 nodes. A total of 1200 data points were collected in the course of these experiments. The 60 data points which corresponded to the shortest simulation time for each circuit were used as our training set.

4.1 Parallel or sequential

In this experiment we used our algorithm to classify a circuit as to whether it should be simulated by a parallel or a sequential simulator. Ten circuits were randomly picked from the circuit pool to be used as query circuits while the remaining 50 circuits were used as training examples. We measure the success of our classification by simply dividing the number of correctly predicted circuits, $num - correct$ by the total number of circuits, $num - total$ and call this ratio the parallel sequential accuracy, psa i.e. $psa = num - correct/num - total$.

This set of experiments was conducted 100 times resulting in an average psa of 92%.

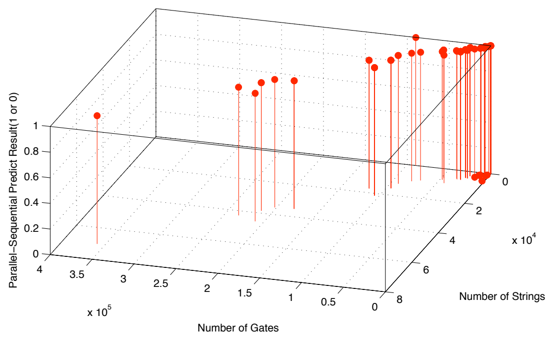

Fig. 4. Parallel or Sequential Predicting Results Distributions

Fig 4 shows the distribution of the parallel-sequential prediction results. From Fig 4, we can see K-NN gives almost 100% correct prediction result for large circuits. The points which were not predicted are crowded in a small gate-number, small string-number corner. To see why large size circuits give better prediction results, we checked the training data set. Fig 5 shows the number of nodes that are labeled for training examples. From Fig 5, we can see that the parallel version of XTW has consistently better performance than sequential ones for circuits have large number of gates and large number of strings. Thus, the K-NN engine generates almost 100% correct results in using parallel or sequential simulator for large size circuits.

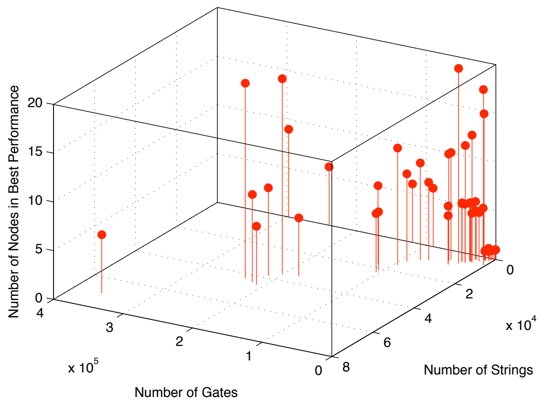

Fig. 5. Distribution of Number of Nodes Generating Best Performance

4.2 Predicting the Number of Nodes

The same experimental set up was used as in the first set of experiments. Ten circuits were randomly picked from the circuits to be used as verification samples and the rest of the circuits were used as training examples. The Euclidean distance based on 3 attributes (the number of gates, the number of partition strings and the number of module instances) was used to select the nearest neighbor.

A query circuit is labeled with the number of nodes that the nearest training example used. If the number of nodes exactly matches the number of node that generates the shortest simulation time of the verification circuit, the predicted result is true. The parallel node number predict accuracy is denoted by pna, the number of correctly predicted values is denoted by cpv, the number of total query circuits is denoted by tqc. pna is defined as: $pna = cpv/tqc$. After 100 experiment runs, the average parallel node number predict accuracy was 53%.

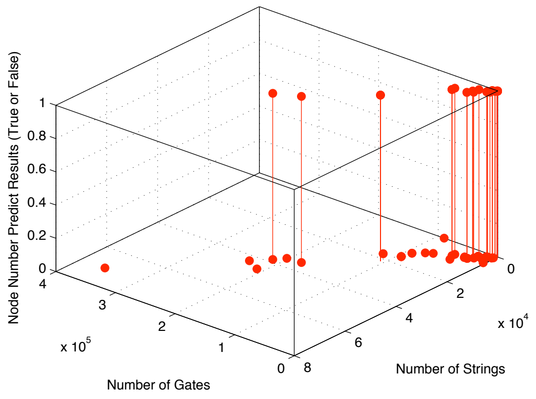

Fig. 6. Distribution of Predicting Number of Nodes

Fig 6 shows the distribution on the results for predicting the number of nodes. The correct result is marked 1 while the incorrect result is marked as 0. From Fig 6, we can see that K-NN gives better predicting rate for small size of circuits. Compare Fig 6 and Fig 5, we can see there are more circuits crowded in small gate-number, small string-number area, this helps the K-NN to generate better results in predicting the best number of nodes for circuit simulation.

4.3 Performance Evaluation

Fig 7 compares the number of nodes that predicted for each query with the number of nodes that labeled to generate best performance. Although K-NN generates a 53% exact match, Fig 7 shows that most predicted number of nodes are close to the best number of nodes. It is interesting to see how well the simulation will use the K-NN predicted hardware option even if it is not an exact match.

In this experiment, we compared the simulation time generated by using predicted hardware options with the best simulation time. We made use of a metric, the performance-prediction efficiency denoted as ppe. The best simulation time is denoted as bst. The simulation time generated by using predicted hardware option is denoted by pst. ppe is defined by: $ppe = bst/pst$

Fig 8 depicts the distribution of ppe. The average performance prediction efficiency was 83%.

4.4 Feature Scaling

The accuracy of the k-NN algorithm is sensitive to the feature selecting and scaling. It can be severely degraded by

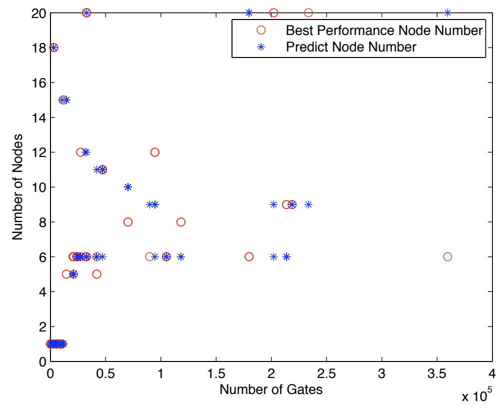

Fig. 7. Compare the best performance node number with the node number predicted

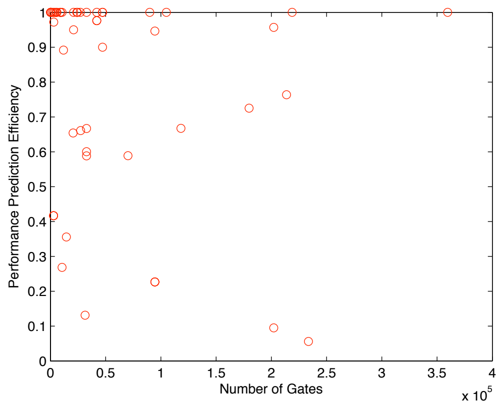

Fig. 8. Performance Prediction Efficiency

the presence of irrelevant features or if the feature scales are not consistent with their importance.

Much previous research effort has been contributed into selecting or scaling features to improve K-NN classification, such as using evolutionary algorithms. In this research, we use a primitive method to explore if scaling features can help in parallel circuit simulation scheduling. We pick a set of weight as (1, 0.1,0.01,0.001,0.0001), then apply these weights to 3 attributes one by one. Examining all of the possible 125 weight combinations, we found that the weight combination which using 0.1 on the number of gates, 0.1 on the number of partition strings and 0.001 on the number of modules resulted a best psa at 99%. This result improves from 92% when no features scaling applied. Fig 9 shows the Parallel-Sequential Predict Accuracy changes as the weights on the number of gates and the number of strings. In Fig 9, the weight of module number is set as 0.001.

Examining all of the possible 125 weight combinations, we found that the weight combination which using 0.01 on the number of gates, 0.1 on the number of partition strings and 0.01 on the number of modules resulted a best Exact-Node-Number-Match-Rate at 61%. This result improves from 53% when no features scaling applied. Fig 10 shows the accuracy on predicting exact match number of nodes exactly match changes as the weights on the number of gates and the number of strings. In Fig 10, the weight of module number is set as 0.01.

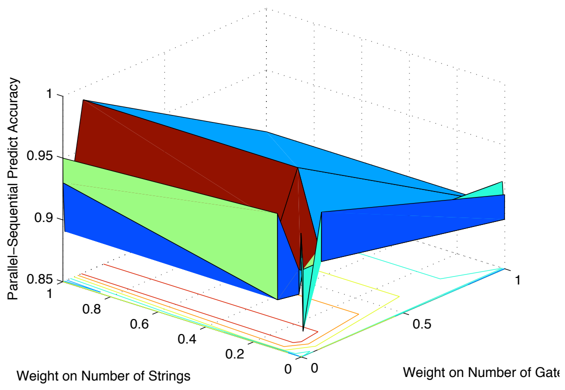

Fig. 9. Scaling Features to improve Parallel Sequential Prediction

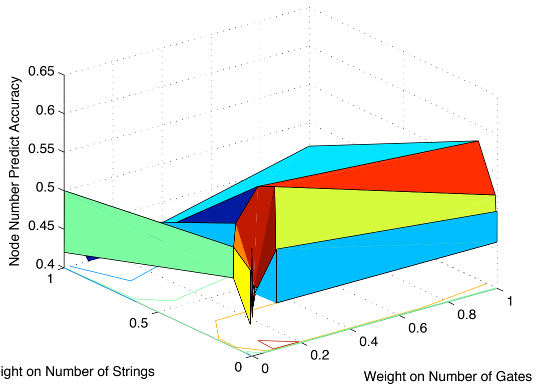

Fig. 10. Scaling Features to improve Number of Nodes Prediction

Examining all of the possible 125 weight combinations, we found that the weight combination which using 0.01 on the number of gates, 0.001 on the number of partition strings and 1 on the number of modules resulted a best performance-predict-efficiency at 89.2%. This result improves from 83% when no features scaling applied. Fig 11 shows the performance-predict-efficiency changes as the weights on the number of gates and the number of strings. In Fig 10, the weight of module number is set as 1.

From above results, we can see that feature-scaling has the potential to improve K-NN classification accuracy in PDES. A more sophisticated feature scaling mechanism , such as, using evolutionary algorithms may further improve the K-NN performance.

4.5 Two-Phase K-NN Optimization

In order to find the nearest neighbor, the K-NN algorithm must compute the distances from the query circuit to all of

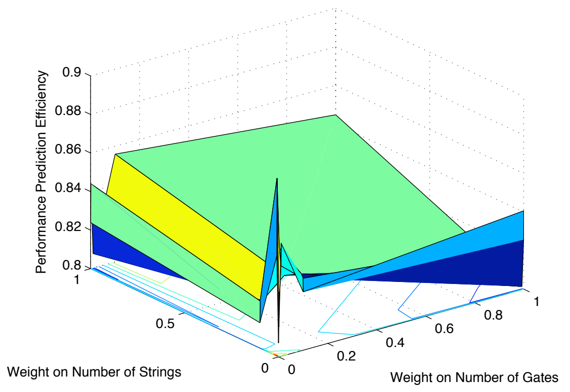

Fig. 11. Scaling Features to improve performance-predict-efficiency

the (stored) training vectors. This is very computationally intensive, especially when the size of the training set is large. Much previous research has been done to reduce this cost including the use of Kd-trees and locality sensitive hashing (LSH). In this paper, a simple approach, 2-phase K-NN, is proposed in order to reduce the computational cost of K-NN for PDES scheduling. The 2-phase mechanism works as follows: the algorithm starts by using only one feature , the number of gates, to do a K-NN search. The K nearest data points are selected in order to be used as training examples for a second phase. In the second phase a normal K-NN iteration is utilized in order to find 1 nearest neighbor among the K data points. The motivation of the 2-phase approach is to use the low cost one dimensional search (O(log(n))) in order to reduce the size of search space. It then does a K-NN in the second phase on a reduced search space. For example, if there are one million training examples we can set K=1000 in the first phase. Then in the second phase, K-NN only needs to search among 1000 data points. The 2-phase algorithm is certain to reduce the computing cost, however it is necessary to (experimentally) assess the accuracy of the classification. In these experiments, K was set to 10 in the first phase and optimized feature weights were applied. The average parallel-sequential accuracy using 2-phase K-NN was 92% (without the 2-phase optimization the peak value was 99%). The average pna was 46% (without the 2-phase optimization the peak value was 61%) . The average performance-predict-efficiency was 83% (without the 2-phase optimization he peak value was 89.2%). From these results we can see that the 2-phase K-NN algorithm reduced the K-NN computing cost and produced a comparable accuracy.

4.6 *Adding New Training Data*

In the quick paced circuit design industry, it will constantly be necessary to add new training points so that the training accounts for new circuits. In a typical circuit design environment, there could be hundreds or thousands of hardware options available. Adding training data by utilizing all of the hardware options is not practical. In this section, we propose a new method, K-data-training, in order to reduce the cost of adding new training data.

We first create a small training set. In the initial training phase, a set of typical circuits are simulated on all of the available hardware options in order to discover the best performance points. After this initial training phase, when a new circuit needs to be added to the training set, we use a K-NN search to find K nearest neighbors. Then, the new circuit is simulated with these K options one by one. The best performance option among these K results is stored in the training set.

Experiments were conducted to verify how well the k-data-training method worked. The same experimental set up was used as in our previous experiments. Ten circuits were randomly picked from the circuits to be used as verification samples and the rest of the circuits were used as training examples. K was set to 5. The Euclidean distance based on 3 attributes (the number of gates, the number of partition strings and the number of module instances) was used to select the 5 nearest neighbors.

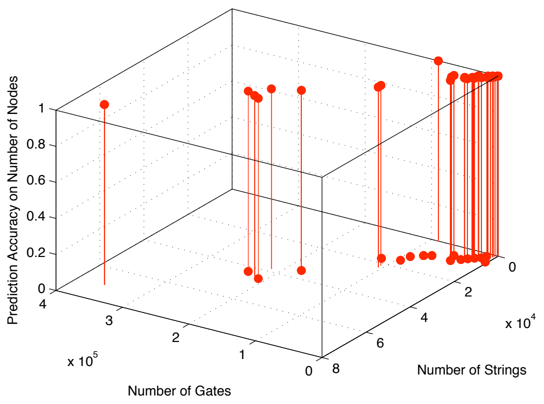

Fig. 12. K-data-training improves Prediction Accuracy on Number of Nodes

The experimental results show that using K-data-training method, the average exact match node number predict accuracy was 75.5% , the average performance-predict-efficiency was 93%. Fig 12 shows the prediction results for the number of nodes. Comparing with Fig 6, the K-data-training method greatly improved the accuracy of predicting the number of nodes which generate the best performance. In a circuit design environment, a circuit normally need to be simulated many times. The K-data-training method may also apply for a circuit whether it needs to be added to a training set or not.

5 Conclusion

We have presented, in this paper, a machine learning algorithm for determining whether a gate level circuit simulation should be run sequentially or on a parallel platform. In the event that it is worthwhile running it on a parallel machine, the algorithm determines the number of nodes on which to execute the simulation. To our knowledge, this is the first attempt to decide this issue. The algorithm is based on the K Nearest Neighbor algorithm and relies on feature vectors to classify the simulations as being either sequential or parallel and as to how many nodes to employ if they fall in the parallel category. A multi-path optimization technique was used to decrease the complexity of the distance computations We used 60 benchmark circuits in our experiments. Of these circuits, 50 were used for training purposes and 10 were used for classification. The hardware we used consisted of a Sun server and a cluster of 20 Linux machines. Our algorithm made the right choice between sequential and parallel simulation 97% of the time. It chose the number of nodes corresponding to the minimum simulation time 53% of the time, but was always within 17% of the minimum simulation time. These results were then greatly improved by a feature scaling algorithm which picked the correct number of nodes 61% of the time and was, on the average, within 11% of the minimum execution time An 2-phase K-NN optimization technique was presented which reduced the computing cost of the K-NN algorithm, but still gave comparable results to those produced by our algorithm. Finally a K-data-training algorithm for reducing the computational cost of adding new training data. It did so while improving the performance of the original algorithm. While our results are promising, it is still desirable to explore other attributes for feature classification and to explore other techniques for scaling feature values and for determining feature correlations.

The results obtained in this paper are certainly promising. Perhaps the most important point is that the general methodology is applicable to virtually any problem in parallel simulation.

References

[1] K.-H. Chang, H.-W. Wang, Y.-J. Yeh, and S.-Y. Kuo. Automatic partitioner for distributed parallel logic simulation. 2004.

[2] Chau-Shen Chen, Ting Ting Hwang, and CL Liu. Architecture driven circuit partitioning. pages 383–389, 2001.

[3] Jong-Sheng Cherng, Sao-Jie Chen, Chia-Chun Tsai, and Jan-Ming Ho. An efficient two-level partitioning algorithm for vlsi circuits. pages 69–72, 1999.

[4] H.Avril and C.Tropper. Scalable clustered time warp and logic simulation. *VLSI Design, Special Issue on Current Advances in Logic Simulation, Gordon-Breach, vol.19, no.3, lpp.291-313*, 1999.

[5] M.D Hutton, J.S Rose, and D.G. Corneil. Automatic generation of synthetic sequential benchmark circuits. pages 928–940, 2002.

[6] D. Jefferson and H. Sowizral. Fast concurrent simulation using the time warp mechanism, part ii: Global control. Technical Report TR-83-204, Rand Corporation, 1983.

[7] Hong Kyu Kim. Parallel logic simulation of digital circuits.

[8] Lijun Li and Carl Tropper. A design-driven partitioning algorithm for distributed verilog simulation. pages 211–218, 2007.

[9] Tun Li, Yang Guo, , and Si-Kun Li. Design and implementation of a parallel verilog simulator: Pvsim. pages 173–180, 2004.

[10] F. Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, pages 423–434, 1993.

[11] Carl Tropper Qing Xu. Xtw, parallel and distributed logic simulation. *19th Principles of Advanced and Distributed Simulation*, pages 181–188, 2005.

[12] Carl Tropper Qing Xu. Towards large scale optimistic vlsi simulation. *Journal of Simulation Modelling Practice and Theory*, 14(6):695–711, 2006.

[13] M.Bailey J.Briner R.Chamberlain. Parallel logic simulation of vlsi systems. *ACM Computing Surveys, vol.26, no.3, Sept. 1994, pp. 255-295*, 1994.

[14] D.E. Martin R.Radhakrishnan D.M.Rao M. Chetlur K. Subramani and P.A. Wilsey. Analysis and simulation of mixed-technology vlsi systems. *Journal of Parallel and Distributed Computing*, 62(3):468–493, 2002.

[15] S.J. Turner and M.Q. XU. Performance evaluation of the bounded time warp algorithm. *6th Workshop on Parallel and Distributed Simulation*, pages 117–126, 1992.

[16] P. Wilsey and A. Palaniswamy. Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation, 1994.