2013 22nd Australian Conference on Software Engineering

An Empirical Experiment on Analogy-based Software Cost Estimation with CUDA Framework

Passakorn Phannachitta*, Jacky Keung†, and Ken-ichi Matsumoto*
*Graduate School of Information Science, Nara Institute of Science and Technology, Japan
Email: {phannachitta-p, matumoto}@is.naist.jp
†Department of Computer Science, City University of Hong Kong, China
Email: Jacky.Keung@cityu.edu.hk

***Abstract*—The success of estimating software project costs using analog-based reasoning has been noticeable for over a decade. The estimation accuracy is heavily depends on different heuristic methods to selecting the best feature subsets and a suitable set of similar projects from the repository. A complete search of all possible combinations may not be feasible due to insufficient computational resources for such a large search space. In this work, the problem is revisited, and we propose a novel algorithm tailored for analogy-based software cost estimation utilizing the latest CUDA computing framework to enable estimation with large project datasets. We demonstrated the use of the proposed distributed algorithm executed on graphic processing units (GPU), which has a different architecture suitable for compute-intensive problems. The method has been evaluated using 11 real-world datasets from the PROMISE repository. Results shows that the proposed ABE-CUDA approach is able to produce the best project cost estimates by determining the best feature subsets and the most suitable number of analogous projects for estimation, significantly improves the overall feature search time and prediction accuracy for software cost estimation. More importantly, the optimized estimation result can be used as a baseline benchmark to compare with other sophisticated analogy-based methods for software cost estimation.**

***Keywords*-Software Cost Estimation, Analogy, CUDA, Empirical Experiments**

I. Introduction

The essential hypothesis of software development effort estimation using analogy-based estimation (ABE) is that similar software projects with similar characteristics will require similar efforts to produce [1]. While the key idea and processes of analogy-based reasoning sound straightforward, it is an approach that has received considerable attention and has been frequently attempted in software engineering [2]. It has proven to be a viable alternative method in terms of accuracy when compared with other approaches such as regression [3].

Analogy-based reasoning requires exploring a large number of different combinations of variable subsets and a suitable number of the nearest neighbouring projects to base estimate [1], in short, a large search space, which takes considerable amount of time to complete and sufficiently powerful computational architectures were not available. The required exhaustive search to retrieve analogous projects were replaced with heuristic search algorithms in a limited search space, resulting different conclusions depending on the heuristic search algorithm applied, and that the prediction accuracy may not be optimum. However, researchers were aware that enhancing the analogy-based reasoning approach by broadening the search space to include more past knowledge and solutions should be considered when judging the capability of this approach.

The main goal here is to determine the best subset combinations together with the best number of neighbouring reference projects which provides better estimation accuracy. To achieve that, a novel algorithm, ABE-CUDA is proposed. The algorithm was supported by the CUDA computing framework, which has been proven as an alternative computational paradigm providing high performance even when the deployment is on a single machine [4]. In the evaluations, we used 11 real-world datasets form the PROMISE software engineering data repository [5], which is commonly used by the software engineering researchers to address problems related to software effort estimation. The results demonstrate the best accuracy derived from the best combination subsets that is otherwise impossible to achieve using existing heuristic-based algorithms. Moreover, the sufficient computing resource availability can conclude one of the most inconclusive issues in ABE discussed about the best number of k value of the utilized k nearest neighbor (i.e. KNN), and the best project feature subsets to use.

The rest of the paper is organized as following. In Section II, we briefly overview analogy-based software cost estimation and the baseline ABE algorithm for evaluation. In Section III, we explain how we apply the algorithm utilizing CUDA computing framework, and include the optimization techniques applied. For the experiments and results, see Section IV and V respectively. Next, Section VI provides detailed discussions on computation performance and deployability. Section VII provides related works including GPU computing with CUDA framework, and a review on the challenges of software engineering problems with high performance computing (HPC). Finally, we conclude the paper and address possible future works in Section VIII.

II. Software Cost Estimation by Analogy (ABE)

Based on the notion that similar projects with respect to similar characteristics may require similar development costs, Shepperd et. al [1] explained the processes using the R^4 model proposed by Aamodt and Plaza [6] as follows. First is to initialize a new problem as a target case. Descriptions of a new problem are then divided into two parts, namely the description

978-0-7695-4995-8/13 $26.00 © 2013 IEEE
DOI 10.1109/ASWEC.2013.28

part and the solution part. The description part consists of a vector of features that describe the case, and the solution part holds the solution for the case. Next is to retrieve information from past projects. The similarity between past projects and the new problem is measured by comparing the description parts of the new case and the existing cases using a similarity measure. Finally, an aggregation of the solution parts becomes a solution for the new case. It is more commonly known as case-based reasoning (CBR) [6] .

From these processes, the computational modules mainly concern the similarity calculation between cases, and the aggregation of the solution parts from the past projects. The similarity between the target case and the existing cases is determined by a similarity measure. The most frequently used is an extension of Euclidean distance called the Heterogeneous Euclidean-Overlap Metric ($HEOM$) [7], which is an effective method for calculating the distances between vectors containing continuous and categorical values. It has also been adopted in Shepperd et. al [1]. The distance between a project P_i and P_j having f features is defined as:

$$dist(P_i, P_j) = \sqrt{\sum_{k=0}^{f} \delta(P_{ik}, P_{jk})^2} \tag{1}$$

$\delta(P_{ik}, P_{jk})$ defines the distance between P_i and P_j observed with only one feature k :

$$\delta(P_{ik}, P_{jk}) = \begin{cases} \frac{|P_{ik} - P_{jk}|}{range_k} & \text{k is continuous} \\ overlap(P_{ik}, P_{jk}) & \text{k is categorical} \end{cases} \tag{2}$$

The value $range_k$ is used to normalize the feature k into the range "0 to 1", and is defined as $range_k = max_k - min_k$ (i.e. the maximum and minimum values of ${P^T}_k$. The function $overlap$ is defined as:

$$overlap(P_{ik}, P_{jk}) = \begin{cases} 0 & P_{ik} = P_{jk} \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

The function assigns the maximum distance to the feature values that are the same and the minimum distance for the feature values that are different. To sum up, the use of $HEOM$ allows continuous and categorical project features to be reasonably combined in the distance calculation. Nevertheless, there is a hidden step consuming the most computational resources that have not yet been mentioned. As the description parts are composed in a feature vector, all the features are not necessarily meaningful. Some features may be dependent features while others may not be helpful and may create noise. As a result, those unnecessary features degrade the estimation accuracy unless only the most helpful subset of features is used, which requires a look up. However, to solve the subset feature selection problem enormously increases the complexity of the problem, as it is an optimum subset search problem with the overall complexity scaled to $\mathcal{O}(2^{\mathrm{n}})$ in time-complexity notation, where n is the number of features.

To select the most helpful subset features depends on the performance metric measuring the prediction accuracy. One of the most widely used accuracy measurements for either easy benchmarking or simple comparing the results against with many other approach is magnitude relative error (MRE) [8] which is defined as:

$$MRE_i = \frac{|Actual_i - Predicted_i|}{Actual_i} \tag{4}$$

Where $Actual_i$ is the actual cost (i.e. solution) of P_i and $Predicted_i$ denotes the estimated cost of P_i. The overall accuracy regarding a selection feature subset comes from the average error of all cases' prediction results which is defined as the mean magnitude relative error ($MMRE$):

$$MMRE = \frac{1}{n} \sum_{i=0}^{n} MRE_i \tag{5}$$

The best feature subset is the one that produces the best accuracy or with minimum errors (i.e. gives the minimum $MMRE$). After the best feature subset is found, the similarity calculation in the R^4 stages will consider only the features in the best feature set. There are other evaluation criteria exist, the choice of $MMRE$ is known for its common use for a generalize measurement of benchmarking effort estimation problems. We opted $MMRE$ in the experiment to demonstrate the ability of our proposed framework when comparing against other approaches whereas our approach can delivers either potentially accuracy gain and accelerated speed up.

A. The Baseline Algorithm

The principal idea of the baseline algorithm mainly considers the scope and lifetime of the examined cases. Difference from the conventional methods such as the ANGEL cost estimation tool [9], the proposed ABE-CUDA algorithm can eliminate the redundant calculation steps that compute the distances between each pair of cases scoping all the possible k value from the KNN in regard to one subset of features (i.e. finding the best subset with the best-k value).

1) ***Subset Generation****:* Although a subset generation task can be simply computed using recurrence, it is rather time consuming and the complexity grows fast. Furthermore, the generated subset is always reused after the deployment, so that all the subsets representation were decided as a one-time pre-calculation.

Each instance of a subset is represented as a vector encoded in a binary string. When input data set consisting of the feature set F of fn members, F is written as a vector $(f_1, f_2, f_3, ..., f_{2^{fn}})$. For example, when F consists of 3 features (i.e.fn is equal to 3), F is represented as vector $(f_1, f_2, f_3, ...f_8)$, and the encoded subset strings are $\{$000,001,010,011,100,101,110,111$\}$. 0 and 1 represent selected and non-selected features respectively.

2) ***Distance Matrix Construction****:* A distance matrix is constructed from input dataset once per an encoded feature subset f. Let $D_{cn \times fn}$ denotes a matrix representing an input dataset which has cn cases and fn features. $M_{cn \times cn}$ denotes the distance matrix consisting of cn rows and cn columns. In the matrix M, each row i and column j stores the distance between $case_i$ and $case_j$ (i.e. or D_i and D_j form $D_{cn \times fn}$).

The distance value stored M_{ij} is calculated by the distance function following Equation 1.

After all the distances in regard to f is calculated and filled into M, M is then transformed into a fast distance-lookup table named matrix M'. The transformation is done by row-wise sorting of M in ascending order, so that in the further step, the sorted value can be directly look-up the first k columns in M' when a number of k neighbors is examined. Additionally, to make the sorted result consistent and applicable, destination case id (j) is augmented with its distance and will be sorted accordingly. Finally, an instance stored in M'_{ij} becomes a tuple $(dist(D_i, D_j), j)$.

*3) **Subset Optimization**:* The best feature subset is the one that produces the minimum $MMRE$ which is a performance metric that its calculation follows the Equation 5. Further than finding the best subset, the best-k neighbors in regard to the examined feature subset are also considered. For this case, the $Predicted_i$ cost in Equation 4 is derived from an aggregation function [9]. In the following example, an average sum is considered. It is one of the most widely used method to aggregate the estimation cost.

As described in Equation 5, $MMRE$ is the summation of MRE regardless to any $case_i$. Since the best feature subset is the one that produces the minimum $MMRE$, all of the $MMRE$ are calculated and compared from each feature subset f. Let R denotes the final result set from the task. R is a triplet holding values $(MinmumMMRE, Best_{set}, Best_k)$. Algorithm 1 explains the entire baseline algorithm in steps.

Algorithm 1 The baseline algorithm

```
1:  D ← read input dataset
2:  tf ← read target feature index
3:  R ← (∞, nil, nil)
4:  for all f in F do
5:      construct M_{cn×cn} from f
6:      M'_{cn×cn} ← row-wise sort M_{cn×cn}
7:      for K = 1 → cn do
8:        fMMRE ← 0
9:        Predicted_i ← 0
10:       for i = 0 → cn − 1 do
11:         Actual_i ← D_{i tf}
12:         for k = 1 → K do
13:           (dis, idx) ← M'_{targetfeature k}
14:           Predicted_i ← Predicted_i + D_{idx tf}
15:         end for
16:         fMMRE ← fMMRE + |Actual_i − Predicted_i/K| / Actual_i
17:     end for
18:     if (1/cn) fMMRE < MinmumMMRE then
19:         R ← ((1/cn) fMMRE, f, K)
20:     end if
21:   end for
22: end for
```

At the first 3 lines of Algorithm 1, the input dataset, the input variable and the result variable are retrieved and initialized respectively. The for-loop between Line 4 and 22 iterates all the possible fn subset features in the feature vector F. In each iteration, An M matrix in regard to f is constructed and row-wise sorted as shown in Line 5 and 6. Next, the for-loop between Line 7 and 21 examine each of k variables to find the best-k neighbors that provide the minimum $MMRE$. The $fMMRE$ variable is a temp variable holding each calculated $MMRE$. The for-loop between line 10 and 17 and its inner for-loop calculate an $MMRE$ in regard to an examining k variable with feature subset f following Equation 5. The calculated $MMRE$ value is the currently minimum $MMRE$, the examining f, k, and the $MMRE$ itself are the temporary result set. At last, the final result set is held in the R variable, and will become the result of this baseline algorithm.

III. Analogy-based Estimation with CUDA

To optimize a task to be executed in advanced parallel platforms such as GPU requires much attentions on load balancing and coupling to achieve maximum efficiency. For example, doubling the computing resources will not immediately result in twice the computational power. There are a number of important factors to consider, such as the partitioning scheme and communication overheads [10]. The main benefit from coupling minimization is that it will enable a load-balancing scheduler that helps to parallel tasks and improves efficiency.

The ABE-CUDA algorithm adopted from the baseline algorithm is the main computing engine. Three subtasks from the baseline increase into five. Except the Subset Generation task, which is a preprocess, four functions in the major modules are replaced with CUDA kernels. The replaced functions are $Distance$ calculation, $Row-wise$ sorting, $Predicted-value$ calculation and $MMRE$ calculation. However, for a better understanding, the detailed explanation for the modules are kept in the same abstract view of the baseline algorithm as distance matrix construction and feature subset optimization.

A. ABE-CUDA : Distance Matrix Construction

Distance matrix construction consists of two major methods, the distance calculation from subsets represent vector f and the row-wise distance sortings that are in accenting order. The distance calculations utilize GPU simply using a traditional loop unrolling technique [11]. This technique unfolds steps in a loop-like process and compute each of them in parallel. For example, the distances between any pair of cases are calculated under an iterated matching scheme between all pair of cases in a sequential order. To take the process to GPU, the iteration is unrolled and computed simultaneously using the GPU compute cores.

On the other hand, the M' matrix transformation involves with sorting that requires a technique more sophisticated than the simple loop unrolling technique to gain performance. The row-wise sorting for M' is different from the well-discussed sorting algorithm for parallel such as Bitonic sort [12] that the M' transformation sorts cn arrays having cn elements concurrently. The difference in those well-discussed sorting algorithm aims to sort an exceptional large single array quickly. The chosen sorting method for the M' transformation is to utilize the GPU shared-memory space, and let cn sorting instances sort cn arrays in parallel using a primitive sorting algorithm.

The advantage of using primitive sorting algorithm is that a very low allocation overhead when comparing with more complex algorithms. It is especially the case when the number of the elements to be sorted are small enough. Algorithm 2 further explains the distance matrix construction part of the ABE-CUDA algorithm.

Algorithm 2 ABE-CUDA : Distance matrix construction

1: **function** CONSTRUCTDISTANCEMATRIX()
2: $D, cn, fn \leftarrow$ read the input
3: Init $M \leftarrow (0, column_{index})_{cn \times cn}$in parallel
4: Move M, D, cn, fn to the GPU memory
5: $M \leftarrow$ compute each $dist(D_i, D_j)$ in D in parallel
6: $M' \leftarrow$ sort M by M_i in ascending order in parallel
7: Move M' back to host memory
8: Free the GPU memory
9: **end function**

The changes from the baseline algorithm are the additional memory allocation and data transfer between host and GPU in line 4, 7, and 8. These are always required when a task is deployed to run in a GPU. For the two modified core functions, both of them are located in line 5 and 6 respectively.

B. ABE-CUDA : Subset Optimization

The two remaining tasks are the $Predicted-value$ calculation and the $MMRE$ calculation. To deploy the $Predicted-value$ calculation in GPU, a predicted-values caching matrix $P_{cn \times k}$ is defined to cache the intermediate $Predict_i$ result on all the examined k values when all of them are simultaneously examined in GPU. Each P_{ij} stores the prediction value of $case_i$.

The $MMRE$ values regard to the scope of k are also expanded and calculated for each of them in parallel. This step is to find the minimum values from the summation of differences between each column of $P_{cn \times K}$ (the case dimension) and target field in input dataset D following the Equation 5. At last, the minimum $MMRE$ is reduced from the array of $MMRE$ values in the GPU memory and store in the final result set R. Algorithm 3 combines all the core functions as the algorithm executes on GPU.

Algorithm 3 The ABE-CUDA Algorithm

1: Read $D \leftarrow$ input dataset
2: Move D to the GPU memory
3: Init R
4: **for all** f **in** F **do**
5: $M' \leftarrow$ Execute ConstructDistanceMatrix()
6: $P \leftarrow$ Predict each P_{ij} using $M'_{i,0..k}$ in parallel
7: $R \leftarrow$ Compute accuracy on each k between $P_{k,i}$ and D in parallel
8: Free M', P in GPU memory
9: **end for**
10: Move R to host memory
11: Free the rest variable in the GPU memory

The four major modules are adopted into GPU kernels and are executed continuously between line 5 and 7, to decrease complexity. Although, each module has different levels of nested loop, its low coupling design enables them to be unrolled independently and yields a higher performance from the GPU computing scheme. Although the proposed technique is able to gain performance, there is also a trade-off factor, it is the data manipulation. As shown in Line 2, 8, 10, and 11, memory allocation and data transfer is presented. The data management is one of the most important factor to judge if a task is appropriate for GPU. For example if a task needs a massive memory transfer, but it has only few operations manipulating over the data, the memory transfer time will dominate the computing time. In these cases, GPU computing will be inferior to other parallel computing schemes. For the purpose of the study, we assume that the memory management time factor can be neglect. We will include that discussion in the following sections.

Figure 1 is an example to demonstrate how the ABE-CUDA algorithm works. Assuming that the input dataset has 8 cases, and the observing k value is 3. Figure 1 shows the data representation from the M matrix is sorted until all the $MMRE$, regardless of all the observing k value, are calculated.

The left-most column represents an instance of the Actual effort field which is located in the solution part of the input dataset. The second column from the left presents only index fields in M' which are the row-wise distances sorted in ascending order. In this example the k is reduced to 3, only the top 3 nearest neighbors which are held in the three left most columns in M', are being observed. Next, prediction matrix P presents the values after the calculated prediction values were done. For example, 467 in the top-right corner comes from $\frac{800+200+400}{3}$ following Equation 5. The first column to the right shows the minimized MMRE in the result vector R as 1.034, and the best-k is equal to 2.

IV. EXPERIMENTS

A. PROMISE datasets

Table I summarizes the real world datasets that were used in our experiments. All 11 datasets were obtained from the PROMISE repository [5], the selected datasets were commonly used by researchers to address software cost estimation problems. In Table I, the second and the third column to the left show the number of cases and features of the datasets. Some of the features should be excluded such as the dependent variables of the solution parts. For example, "Development duration" is a dependent variable of "Man-hour effort". The concluded numbers of features used in experiments are shown in the fourth column. For example, one of four features are excluded from the Telecom1 dataset. The next two columns show the number of classified the type of features; continuous and categorical features. At last, the problem size is classified into three groups using the number of included features. The feature size is considered over the number of cases as it affects the time complexity.

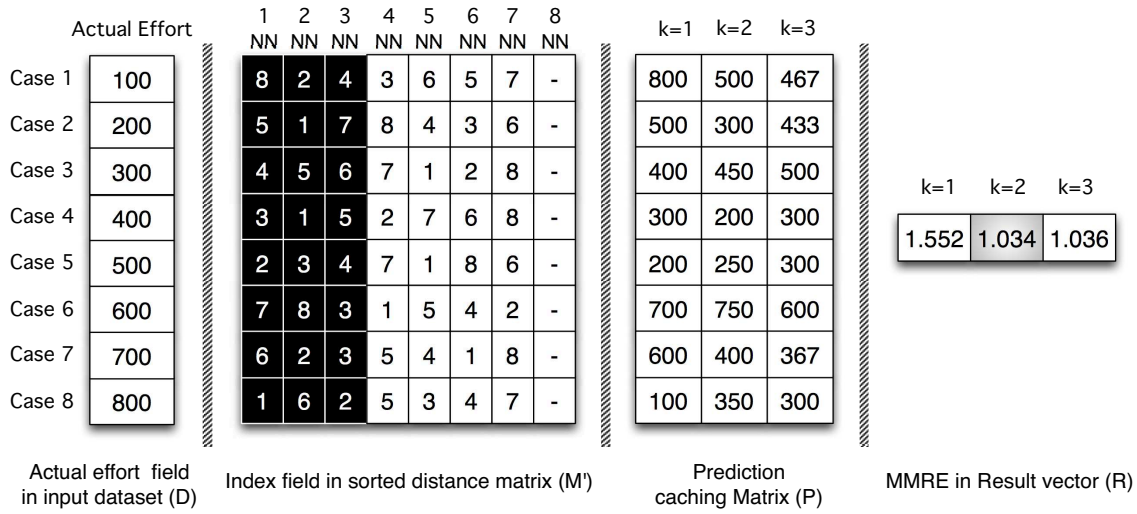


Fig. 1: An example of data representation and workflow given an 8 cases dataset scoping k=3

TABLE I: The 11 PROMISE datasets

Dataset	*Number of projects*	*Number of Features*	*Included features*	*Continuous type*	*Categorical type*	*Classified size*
Telecom1	18	4	3	3	0	Small
Kitchenham	132	10	5	3	2	
Kemerer	15	8	6	4	2	
Albrecht	24	8	8	8	0	
Miyazaki94	48	9	8	8	0	
Desharnais	77	11	10	8	1	Medium
China	499	19	12	10	2	
Usp05-ft	58	15	14	5	9	
Coc81-dem	63	27	25	3	22	Large
Nasa93-dem	93	27	25	3	22	
Maxwell	62	27	26	4	22	

B. Methodology

To evaluate the performance of the ABE-CUDA algorithm. First, we verify the correctness of the ABE-CUDA algorithm implementation against the well-respected analogy-based software effort estimation tool named ANGEL [9]. After the algorithm is verified, we measure the accuracy with an expanded search space using an unbound k value and all possible subset combination. Further than a proof of accuracy, an extended experiment to show the practical advantage is conducted by measuring the actual computation time and improved accuracy.

C. Testbed environments

All the evaluations in this work are deployed on a commodity desktop PC. The machine has an Intel i3 dual-core processor with 8 GB main memory. The deployed GPU device is GeForce GTX 560 Ti, which is only an entry-level graphic device with 384 cores running at 1.64GHz, and with 1GB video memory. For the software and configuration, CentOS 5.5 is chosen as a stable linux distribution, and with the recently released CUDA driver and runtime version 5.0 installed.

V. Results

A. Baseline cross-validation

In the first experiment, the correctness of the ABE-CUDA algorithm was verified by comparing the result with the well-respected ANGEL cost estimation tool [9]. To match with the largest possible k value observed with the ANGEL tool, the k value was limited to 5 in this cross-validation experiment. The comparison is shown in Table II.

TABLE II: Cross validation with ANGEL where k was limited to 5

Dataset	ABE-CUDA MMRE (k=5)	ANGEL MMRE (k=5)	Selected Feature Subsets (Alls are the same)
Telecom1	0.788	0.745	CHANGES
Kitchenham	0.801	0.802	ProjectType | AdjFP
Kemerer	0.723	0.675	KSLOC
Albrecht	0.795	0.763	Output | Inquiry | File | RawFP | AdjFP
Miyazaki94	0.439	0.431	KLOC | EFILE
Desharnais	0.404	0.400	AdjFps | DevEnv
China	1.443	1.441	AFP | INPUT | Enquiry | Interface | Added | Changed | Delete
Usp05-ft	0.315	N/A	DataFile | DataEN | UFP | DBMS | Method
Coc81-dem	1.102	N/A	Rely | Kloc | defects
Nasa93-dem	0.811	N/A	apex | defect
Maxwell	0.587	N/A	T09 | Size

We found unusual results based on the Usp05-ft dataset. Although ANGEL completed calculation on this dataset, the result was considered incorrect that it is different from what we have manually calculated. It is believed a limitation of the

ANGEL tool itself that it may not handle datasets with more 14 features. For the dataset which have features size below 14, the optimum result sets produced by both frameworks are identical, when the $MMRE$ values are presenting slightly different due to different decimal precisions in the two implementations. In addition, we could not obtain the results of the large datasets from ANGEL because it did not terminate within a reasonable timeframe. However, we can confirm the $MMRE$ values by specifying our result set in ANGEL, and the differences are relatively small.

Note that, we have excluded the dependent variables to the ABE solution part from the search as we mentioned earlier. The variables shown in Table II are the selected features, which are useful for prediction. These variables selected are well-recognized in ABE because they are practically estimations in prior using the software techniques, such as IFPUG and COSMIC [13].

B. The ABE-CUDA estimation accuracy

In this experiment, the limited observing k value is unbounded to the number of case size for each dataset. To search for the best $MMRE$ (i.e. minimum $MMRE$) from the best-k at the same time with the best subset combination. The k value, $MMRE$, and the result set are reported in Table III.

TABLE III: The accuracy from the best-k neighbors and the best subset combination

Dataset	Best-k	Minimum MMRE	Most Important Feature Subsets Selected by ABE-CUDA
Telecom1	1	0.423	FILES
Kitchenham	2	0.733	Clientcode | ProjectType | AdjFP
Kemerer	2	0.563	Language | AdjFP
Albrecht	3	0.648	Input | Output | File | AdfFP
Miyazaki94	1	0.396	KLOC | ESCRN
Desharnais	3	0.364	Adj Fps | Dev Env
China	**81**	**1.392**	**AFP | Input | Enquiry | File | Interface | Added | Deleted**
Usp05-ft	1	0.145	DataFile | DataEN | DataOut | Tools | AppExpr | TeamSize | DBMS | AppType
Coc81-dem	2	0.612	Rely | Defect
Nasa93-dem	1	0.515	Pmat | data | time | plex | Kloc | defect
Maxwell	1	0.495	Har | Dba | Ifc | Source | T02 | T05 | T09 | T14 | Size

First of all, the reported best-k value demonstrated more than half of the datasets have a best-k value larger than 1. In such case, it would spend k times the computing effort to search using the available tools such as ANGEL, thus a search of the medium and large dataset seems not to terminate with only the k limited to 1. Next, focus on the best result subset column, the best result subset is independent to the best-k that a simplification by iterating only the k value over the best initial subset combination is not sufficient. The most worth noting result shown in Table III is that the best-k value of 81 from the China dataset. To the best of our knowledge, none of the existing tool that utilize either an exhaustive or a heuristic-base search technique can achieve this accuracy (i.e. 1.392 Minimum MMRE).

The result from searching for the best-k in correspondence with the best subset combination, can be considered the "maximum achievable" prediction performance given the dataset available [14], an important tool for researchers or practitioners to use for benchmarking against other methods developed.

VI. Discussions

This section mainly discusses the $MMRE$ results from our proposed ABE-CUDA and the practical advantage from utilizing GPU with the CUDA computing platform. For some large datasets with higher dimensions, traditional ABE is impractical due to the large amount of data needs to compute, therefore alternatives such as heuristic search algorithms were used but may not be optimum for estimation accuracy. To the best of our knowledge, a consumer-level PC is unable to handle a large project dataset for cost estimation using ABE. As presented in the well-respected ANGEL toolset, a dataset with 25 features did not terminate despite observing only k equals to 1. In this work, with the superior GPU capability and the CUDA framework, we are able to derive ABE-CUDA algorithm to complete the compute-intensive tasks within a reasonable time. we conducted several extended experiments to observe the efficiency in term of speed up, and discuss the deployability of an ABE cost estimation framework using the ABE-CUDA algorithm in this section. The utilization of ABE-CUDA algorithm is a significant improvement to utilize the ABE method for practical usage.

A. Estimation accuracy from ABE-CUDA

We chose MMRE as our main accuracy metric because it has been the de facto criterion to select for benchmarking among the ABE prediction models [8]. Our purpose is to demonstrate the potential technical merit of GPUs utilized ABE framework where the result are clearly comparable with the primitive well-respect toolset (i.e. the ANGEL toolset), we decided not to enchant the algorithm that will deliver a better prediction accuracy and only MMRE is shown in the main result section. As a result the $Minimum - MMRE$ demonstrated in table III from some datasets are appeared too high and may unusable.

Unfortunately, $MMRE$ has been widely criticized for being an unbalanced indicator nowadays, so that the choice of presenting only $MMRE$ become a poor decision [8]. In this discussion section, we present four more accuracy metrics which are commonly discussed as accuracy metrics in ABE; Mean magnitude of relative to the estimate ($MMER$), Mean absolute residual (MAR), Mean balanced relative error ($MBRE$), and Mean inverted balanced relative error ($MIBRE$) [15]. The results are shown in Table IV.

From this table, we can see the best-k are not always optimum. We can often see the best-k larger than 5 in $MBRE$ and $MIBRE$ metrics. Also some of them are especially larger than 40. These results are strongly support how importance of our proposed powerful framework than enables a search for

TABLE IV: Report five accuracy metrics

Dataset	MMRE		MMER		MAR		MBRE		MIBRE	
	Best-k	Best MMRE	Best-k	Best MMER	Best-k	Best MAR	Best-k	Best MBRE	Best-k	Best MIBRE
Telecom1	1	0.423	1	0.495	6	132.943	3	0.0867	7	0.0141
Kitchenham	2	0.733	3	0.560	3	1824.710	**41**	**0.0047**	2	0.0206
Kemerer	2	0.563	2	0.615	2	116.883	7	0.0166	6	0.0169
Albrecht	3	0.648	1	0.402	8	10.397	3	-0.0272	**17**	**-0.0191**
Miyazaki94	1	0.395	3	0.468	3	47.841	**44**	**0.0189**	**31**	**0.0008**
Desharnais77	3	0.364	2	0.368	2	1755.110	7	-0.0083	5	0.0027
China	**81**	**1.392**	**32**	**0.696**	**31**	**2474.360**	2	0.1741	2	0.0065
usp05-ft	1	0.145	1	0.437	1	1.166	**51**	**0.0084**	**41**	**-0.0002**
coc81-dem	2	0.612	4	0.781	3	305.433	**11**	**0.0653**	8	-0.0109
nasa93-dem	1	0.515	7	0.470	3	274.273	**15**	**0.0046**	**24**	**-0.0046**
maxwell	1	0.495	4	0.466	3	3386.420	9	0.0219	10	-0.0013

best-k and best subset consequently and the deployment terminates with the estimation results in an reasonable timeframe.

As each metric suggests different candidate of either best subset or best-k and the question regarding "which model is the best?" are not in our main scope in this work, we would rather like to discuss another aspect of contribution from this result which is all the five different accuracy metrics can be practically tested with our powerful ABE-CUDA framework at the same time with best-k and best subset and the result terminated in an approximate timeframe. In conclusion, the testing result demonstrated in Table IV provides a strong evidence on how the ABE-CUDA could be used in solving ABE-related problems and even some of the data-intensive problems in empirical software engineering research.

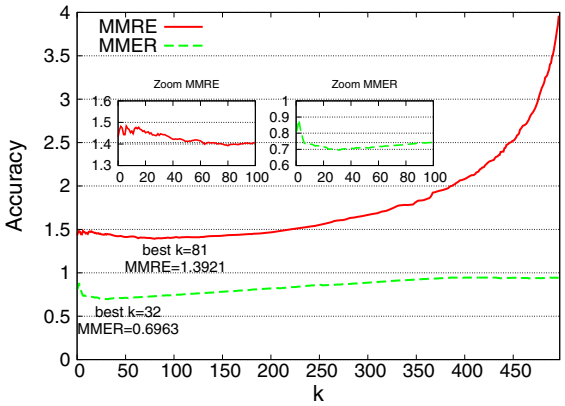


Fig. 2: Iterate all the k neighbors to observe $MMRE$ and $MMER$ in the China dataset

B. High best-k neighbors concluded from the China dataset

As shown in Table III, the best-k neighbors from the China dataset is exceptionally higher than the other datasets when $MMRE$ was the criterion, where as the estimation accuracy achieved from it are not much helpful; moreover, too high errors were also produced from many accuracy metrics comparing to the other datasets. We suggest a possibility of the reason for the issue of high error that the dataset itself should be improved with filtering and extraction technique at the first place such as deploying outliner detection and elimination method to clean the dataset [16].

The issue regarding to high best-k neighbors may also relevant the dataset characteristic; on the other hand, our ABE-CUDA is the only approach that can test and observe all the k neighbors and able to analyze the circumstance based on the k values to provide helpful insight. Figure 2 demonstrated all the iterated best-k value with $MMRE$ and $MMER$ metrics.

The x axis in Figure 2 indicates the iterated k value, and the y axis indicates the accuracy in term of $MMRE$ and $MMER$. Two small boxes inside Figure 2 highlight the area around the best-k values. Observing the zoom-in area from both metrics, the accuracy always fluctuates either it is diverging or converging to an optima. As a result, the best-k for a dataset with different metrics and subsets are unpredictable and make us conclude that exhaustive search is more appropriate to find the optimal estimation solution.

C. Performance in term of computing time and speed up

In this experiment, two ABE cost estimation frameworks were implemented for execution in the host machine with GPU acceleration that follows Algorithm 1 and 3 respectively. Both studies repeated the previous experiment that the k value is unrestricted. The results are measured in computing time, speed up, and the preprocessing time are reported in Table V.

TABLE V: Report computing time and accelerated speed-up

Dataset	CPU (sec)	GPU (sec)	Speed up
Telecom1	0.607	0.639	0.950
Kitchenham	0.762	0.654	1.165
Kemerer	0.662	0.616	1.075
Albrecht	0.704	0.654	1.076
Miyazaki94	0.752	0.690	1.090
Desharnais77	1.822	0.829	2.198
China	864.896	188.426	4.590
Usp05-ft	9.664	2.342	4.127
Coc81-dem	27920.472	5492.788	5.083
Nasa93-dem	**73722.491**	**11039.153**	**6.678**
Maxwell	55979.671	10673.320	5.245

The small datasets reported almost none of speed up achieved that the computing time between and CPU and GPU were approximately the same. However, the better perfor-

mance can be observed by the larger size of the datasets. Speed up was gained by 4.59 times as the maximum for the medium size datasets. The peak performance gained in term of speed was presented in the large dataset, the NASA93-dem at 6.678.

As mentioned above, more performance was usually gained by the larger size of the dataset is probably because the complexity is mostly effected by the size of feature. However, there are an exception in the Nasa93-dem and the China dataset, which their presented magnitude of speed-up were larger than other datasets in the same category size. The exception comes from the extraordinary large size of the cases on the both datasets (e.g. 499 in China and 93 in Nasa93-dem comparing to other datasets) that is large enough to affect the total computing time.

To discuss the maximum speed up gained at 6.678 without a consideration with the computing time, it might not seem importance. However, it became much meaningful when the 6.678 times speed up helped reduce the computing time form 73722.491 seconds to 11039.153 seconds or for approximately a day into a few hours. It clearly helps reduce the computing time into an acceptable waiting time. Furthermore, to gain six time speed up from other traditional parallel computing scheme, at least six computers with a high performance network set up are required. Therefore, the set up and maintenance cost is much higher than employing only a GPU card and using only one machine. Given these points, the ABE-CUDA solution make a cost estimation for such large project possible for an entry-level commodity machine such as laptop equally to the testbed environment that produces the results in Table V.

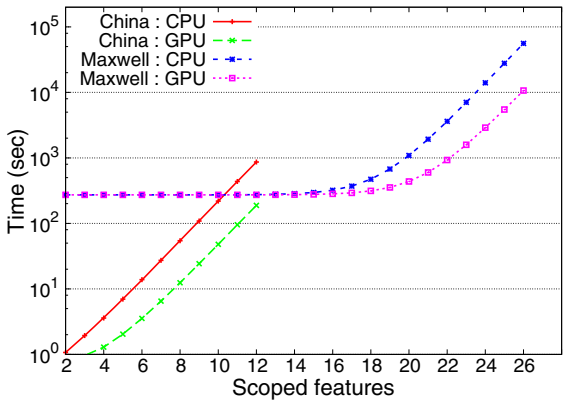
D. Study the relationship between computing performance and the problem size

As shown in the previous experiment, size of the source problem clearly affects the computing performance. Observing the growing size of the problem in a controlled experiment is interesting. It helps the practitioners as well as the researchers to know the the good characteristics of the source problem when the cost estimation task is deployed in GPU.

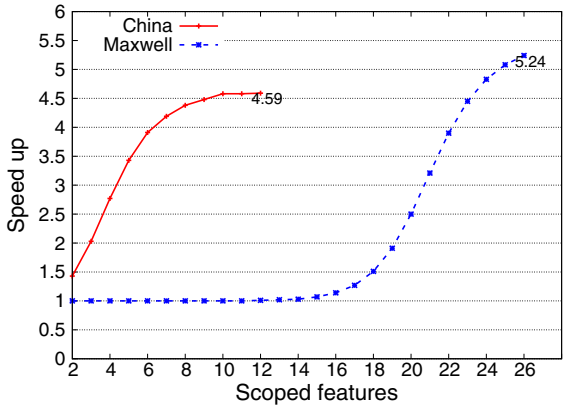
This experiment scoped the problem size as a control variable by increasing the number of included features one by one in China and Maxwell dataset. The two dataset were chosen because they possessed the maximum speed comparing other datasets in the same category. Also China and Maxwell have the most number of cases and features respectively. Figure 3 reports the result in terms of computing time and speed up.

Figure 3(a) reports the total computing time include the subset label generation (i.e. the preprocessing). The x-axis indicates the scoped features size and the y-axis indicates the computing time in log-scale, and a straight line in this coordinate implies a quadratic in the linear coordinate. The log scale is chosen because the problem size grows in exponential (i.e. $\mathcal{O}(2^n)$) after the preprocessing time is dominated as shown in the Figure 3(a) (i.e. after the scoped features equals to 14 in Maxwell dataset).

China is the dataset having largest number of cases with a considerably few number of features. Figure 3(a) demonstrates



(a) Computing time



(b) Speed up

Fig. 3: Iterate the scoped features size in the China and Maxwell dataset

that the preprocessing time for the China has no effect to the overall computing time at all. In contrast to the Maxwell dataset, which is involved with a larger number feature size for the preprocess, but the number of cases is rather small, 270 seconds of the preprocessing time kept dominated the calculation time until the scoped feature size grown to 14.

Figure 3(b) demonstrates accelerated speed up. The x-axis indicates the scoped features size and the y-axis indicates the magnitude of speed up. For a case likes the China dataset, good speed up can be achieved since the number of (scoped) feature sizes is small. However as the problem grows quickly, the speed up becomes saturated pretty fast. As shown in Figure 3(a), the speed up of the feature-size iterated the China dataset increases slowly after the scoped feature size reached 8. Note that the saturated speed up does not imply a performance drop since it is the peaked of the graph. It could be the maximum ratio of the performance gain. On the other hand, in a large problem set such as the Maxwell dataset, the peak performance in speed up is higher than the smaller dataset such as the China dataset, although the speed up increases much slower.

The high magnitude of floating computation for the continuous type features can gain more beneficial advantage from the GPU computing architecture. As shown in Figure 3(b), the accelerated speed up from the China dataset is gained much

faster (i.e. at a small number of the scoped features) comparing to the Maxwell dataset which consists of many categorical type features.

VII. Related works

A. GPU Computing and CUDA programming model

The computing capability of graphic processing unit (GPU) is seen by the realtime computing task such as rendering 3D modeling and high-definition gaming. Since the common graphics tasks are inherently involved with a huge magnitude of floating point calculation to fill millions of pixels in a fragment of second, it arouses an interests in programmable GPU and delivers programming models allow a general instruction directly executed on the GPU. As a result, a general-purpose computation on graphics processing units (GPGPU) solution has recently emerged in a various area of compute-intensive applications, such as linear algebra [17], [18] and bioinformatics [19]. Unfortunately, surveys reported only a few number of study focused on the use of GPGPU in the software engineering domain [20].

Compute Unified Device Architecture or CUDA [21] is one of the most well-known programming model besides AMD brook+ [22] and OpenCL [23]. In the CUDA programming model, it extends the standard ANSI C/C++ with two parallel programming abstractions [11] which are the main reasons that provides acceleration in speed. The two programming abstractions are hierarchal thread blocks and memories, which are inherently involved. In summary, different level of thread manipulations allow a task that has different barrier of synchronization with accessibility to different memory spaces, and the lower level of the hierarchy, the higher bandwidth of the memory space that is much higher than the main memory (i.e. DRAM).

However, there exists a trade-off. A GPU task necessarily requires a co-operation with the host machine CPU and DRAM. In CUDA, both instructions and data are held in DRAM prior to the execution in the GPU, which is a clearly a sturdy constraint in the GPU computing scheme. It especially degrades the overall performance, when the problem demands a larger memory space than the total size of GPU global memory. In this case, multiple data transfer is necessarily executed. The worst case is when a task needs much of memory transfer, but it has only few operations manipulating over the data. In such case, the memory transfer time will dominate the computing time, resulting in ineffective utilization.

B. High performance computing approach to challenge data-intensive software engineering problems

Many challenges in data-intensive software engineering practices are to model complex problems such as estimating software development cost and effort, detecting code clone segments and software testing. These problem require high and sufficient computational resources and infrastructure support.

High performance computing (HPC) approach is well-adopted in the area of clone detection and software testing, where as, to the best of our knowledge, it is not seen in analogy-based software cost estimation (ABE) area. Similar to the clone detection and software teasing area, the problems discussed in ABE are high computing resource requirement. Kirsopp et al suggested statistical model for large ABE problems as the test for all combinations in the exponential-scale search space meant impossible [24]. The successful of HPC approach addressed in the both area attracted our interests to explore the possibility of developing HPC approach into the area of ABE. The following are literatures reports the successfulness in clone detection and software testing area.

Software code clone detection is to detect and analyze duplicate source code fragments, which are harmful for the software maintenance and refactoring. Nowadays, higher number of software projects involve a huge amount of source codes that make a scalable detection framework become necessary. HPC resources, which were difficult to obtain and maintain in the past, are an essential part of effectiveness scalable detection framework. Schugerl et al explored cloud computing and proposed a parallel clone detection algorithm which the scalability has been proved with an analysis on very large datasets [25]. Another proposed algorithm aimed at scalability was presented by Sajnani et al. They used MapReduce programming model, which is a mature HPC programing paradigm, to construct a parallel code clone detection algorithm [26]. The framework inherited many of benefits from the MapReduce programming model such as sophisticated load balancing, data replication, and fault tolerance. Both works presented interesting methods to solve scalability issues in large-scale code clone detection.

Scalable solutions are extensively essential in software testing area. This area has been one of the major applications in Search based software engineering (SBSE) [27]. SBSE is an approach to find an optimum solution to balance multiple software engineering objectives using heuristic search techniques such as genetic algorithm [28]. The computational complexity is very high in SBSE that make it impractical for large scale problems. HPC solution is mostly visited in this area, for example, Geronimo et al recommended Hadoop distributed computing framework and presented a parallel genetic algorithm for the automated generation of test suites [29]. Their result achievement was either scalability proved measuring with branch coverage and a high accelerated speed up improvement at most 50% given an advantage of massive horizontal scaling. They also addressed GPUs computing as an important HPC solution in their work. A work in this area that fully utilized GPUs solution was proposed by Yoo et al [28]. Their study concluded that a GPU-enabled optimization algorithms delivered a faster testing suite that can decrease the total test cost as well as increase the number of testing instances in the same limit of time scope.

VIII. Conclusion and Future work

In this study, we have proposed ABE-CUDA, a novel algorithm for analogy-based software cost estimation. It utilized the CUDA computing platform that accelerates the intensive computational part of the cost estimation problem. For over a decade, analogy-based estimation (i.e. ABE) has been proven to be a useful procedure for software cost estimation despite not being performed at its full predictive power because of the

shortage of resources and lack of a supporting architecture to perform the extensive searches needed. The best contribution from the algorithm evaluation using 11 standard benchmark datasets available from the well-discussed PROMISE repository [5] is, we are able to achieve the best accuracy that is impossible from any of existing tools which utilized either a simplified exhaustive search or a heuristic-base search techniques. An ability to resolve the problem of selecting the best number of k value of the utilized k nearest neighbor (i.e. KNN), and the best subset selection parameter, which is the mainly discussed issue in ABE and was resolved by simplification, yield the best accuracy of the ABE-CUDA. Moreover, the result from the expansion of the search space in the ABE problem would be the "golden standard" showing individual benchmark-standard dataset maximum performance for researchers or practitioners to use for benchmarking their method developed.

In this work, GPU technology helped resolving the shortage of computing resource problems, so that the ABE problem can be deployed without a sacrificed performance in accuracy as presented in the past. The availability of sufficient computing resources would arouse the interested to improve the software cost estimation methods further. In future, we plan to develop a solution utilizing other estimation methods to produce results from either multiple estimation models or more sophisticated and complex techniques, which should be useful to allow software developers to better understand and estimate their project resource requirements.

Acknowledgements

Part of this research was conducted under Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C) (22500028). It was partially supported by the Student Exchange Support Program (Scholarship for Short Term Visit - Short Term Stay) of Japan Student Services Organization(JASSO), the AWS in Education Research Grant award, and the Global Initiatives Program for Promoting Overseas Collaborative Research Toward Graduate Education in Biological Sciences, Nano Science, and Information Technology, Nara Institute of Science and Technology.

The first author places on record his gratitude to associate professor Akito Monden for his insightfully initiation on research collaboration between two research departments, which later become a wise internship opportunity for students, and for his generously administrative support during the internship program.

References

[1] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, 1997.

[2] S. Gupta, G. Sikka, and H. Verma, "Recent methods for software effort estimation by analogy," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 4, pp. 1–5, 2011.

[3] C. Mair and M. Shepperd, "The consistency of empirical comparisons of regression and analogy-based software project cost prediction," in *Proceedings of 2005 International Symposium on Empirical Software Engineering (ISESE)*, 2005, p. 10.

[4] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using cuda," *J. Parallel Distrib. Comput.*, vol. 68, no. 10, pp. 1370–1380, 2008.

[5] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. (2012, Jun) The promise repository of empirical software engineering data. [Online]. Available: http://promisedata.googlecode.com

[6] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches," *AI Commun.*, vol. 7, no. 1, pp. 39–59, 1994.

[7] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *J. Artif. Int. Res.*, vol. 6, no. 1, pp. 1–34, 1997.

[8] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Trans. Softw. Eng.*, vol. 29, no. 11, pp. 985–995, 2003.

[9] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *Proceedings of the 18th international conference on Software engineering (ICSE)*, 1996, pp. 170–178.

[10] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing*, ser. Pearson Education. Addison-Wesley, 2003.

[11] NVIDIA Corporation, "NVIDIA CUDA C programming guide," Oct 2012, version 5.0.

[12] M. Pharr and R. Fernando, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[13] J. J. Cuadrado-Gallego, F. Machado-Piriz, and J. Aroba-Páez, "On the conversion between ifpug and cosmic software functional size units: A theoretical and empirical study," *J. Syst. Softw.*, vol. 81, no. 5, pp. 661–672, 2008.

[14] J. W. Keung, "Theoretical maximum prediction accuracy for analogy-based software cost estimation," in *Proceedings of the 15th Asia-Pacific Software Engineering Conference (APSEC)*, 2008, pp. 495–502.

[15] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Trans. Softw. Eng.*, vol. 31, no. 5, pp. 380–391, 2005.

[16] M. Tsunoda, T. Kakimoto, A. Monden, and K. Matsumoto, "An empirical evaluation of outlier deletion methods for analogy-based cost estimation," in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise)*, 2011, pp. 17:1–17:10.

[17] M. Garland, "Sparse matrix computations on manycore gpu's," in *Proceedings of the 45th annual Design Automation Conference (DAC)*, 2008, pp. 2–6.

[18] V. Volkov and J. W. Demmel, "Benchmarking gpus to tune dense linear algebra," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC '08)*, 2008, pp. 31:1–31:11.

[19] M. Charalambous, P. Trancoso, and A. Stamatakis, "Initial experiences porting a bioinformatics application to a graphics processor," in *Proceedings of the 10th Panhellenic conference on Advances in Informatics (PCI)*, 2005, pp. 415–425.

[20] M. L. Wong, "Parallel multi-objective evolutionary algorithms on graphics processing units," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers (GECCO)*, 2009, pp. 2515–2522.

[21] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[22] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 777–786, 2004.

[23] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *IEEE Des. Test*, vol. 12, no. 3, pp. 66–73, 2010.

[24] C. Kirsopp, M. J. Shepperd, and J. Hart, "Search heuristics, case-based reasoning and software project effort prediction," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2002, pp. 1367–1374.

[25] P. Schugerl, "Scalable clone detection using description logic," in *Proceedings of the 5th International Workshop on Software Clones (IWSC)*, 2011, pp. 47–53.

[26] H. Sajnani, J. Ossher, and C. Lopes, "Parallel code clone detection using mapreduce," in *Proceedings of International Conference on Program Comprehension (ICPC)*, 2012, pp. 261–262.

[27] P. McMinn, "Search-based software test data generation: a survey: Research articles," *Softw. Test. Verif. Reliab.*, vol. 14, no. 2, pp. 105–156, 2004.

[28] S. Yoo, M. Harman, and S. Ur, "Highly scalable multi objective test suite minimization using graphics cards," in *Proceedings of the 3rd international conference on Search based software engineering (SSBSE)*, 2011, pp. 219–236.

[29] L. Di Geronimo, F. Ferrucci, A. Murolo, and F. Sarro, "A parallel genetic algorithm based on hadoop mapreduce for the automatic generation of junit test suites," in *Proceedings of the 5th International Conference on Software Testing, Verification and Validation (ICST)*, 2012, pp. 785–793.