Highly Scalable Multi Objective Test Suite Minimisation Using Graphics Cards

Shin Yoo[1] Mark Harman[1] and Shmuel Ur[2]

[1] University College London
[2] University of Bristol

Abstract. Despite claims of "embarrassing parallelism" for many optimisation algorithms, there has been very little work on exploiting parallelism as a route for SBSE scalability. This is an important oversight because scalability is so often a critical success factor for Software Engineering work. This paper shows how relatively inexpensive General Purpose computing on Graphical Processing Units (GPGPU) can be used to run suitably adapted optimisation algorithms, opening up the possibility of cheap scalability. The paper develops a search based optimisation approach for multi objective regression test optimisation, evaluating it on benchmark problems as well as larger real world problems. The results indicate that speed–ups of over `25x` are possible using widely available standard GPUs. It is also encouraging that the results reveal a statistically strong correlation between larger problem instances and the degree of speed up achieved. This is the first time that GPGPU has been used for SBSE scalability.

1 Introduction

There is a pressing need for scalable solutions to Software Engineering problems. This applies to SBSE work just as much as it does to other aspects of Software Engineering. Scalability is widely regarded as one of the key problems for Software Engineering research and development [1, 2]. Furthermore, throughout its history, lack of scalability has been cited as an important barrier to wider uptake of Software Engineering research [3–5]. Without scalable solutions, potentially valuable Software Engineering innovations may not be fully exploited.

Many search based optimisation techniques, such as evolutionary algorithms are classified as 'embarrassingly parallel' because of their potential for scalability through parallel execution of fitness computations [6]. However, this possibility for significant speed–up (and consequent scalability) has been largely overlooked in the SBSE literature. The first authors to suggest the exploitation of parallel execution were Mitchell et al. [7] who used a distributed architecture to parallelise modularisation through the application of search-based clustering. Subsequently, Mahdavi et al. [8] used a cluster of standard PCs to implement a parallel hill climbing algorithm. More recently, Asadi et al. [9] used a distributed architecture to parallelise a genetic algorithm for the concept location problem.

Of 763 papers on SBSE [10] only these three present results for parallel execution of SBSE. Given the 'embarrassingly parallel' nature of the underlying approach and the need for scalability, it is perhaps surprising that there has not been more work on SBSE parallelisation. One possible historical barrier to wider application of parallel execution has been the high cost of parallel execution architectures and infrastructure. All three previous results cited in the previous paragraph used a cluster of machines to achieve parallelism. While commodity PCs have significantly reduced the cost of such clusters, their management can still be a non-trivial task, restricting the potential availability for individual developers.

Fortunately, recent work [11] has shown how a newly emerging parallelism, originally designed for graphic manipulation, can be exploited for non–graphical tasks using General Purpose computing on Graphical Processing Unit (GPGPU) [12]. Modern graphics hardware provides an affordable means of parallelism: not only the hardware is more affordable than multiple PCs but also the management cost is much smaller than that required for a cluster of PCs because it depends on a single hardware component. GPGPU has been successfully applied to various scientific computations [13, 14]. However, these techniques have never been applied to Search-Based Software Engineering problems and so it remains open as to whether large-scale, affordable speed–up is possible for Software Engineering optimisations using GPGPU to parallelise SBSE.

Fast regression test minimisation is an important problem for practical software testers, particularly where large volumes of testing are required on a tight build schedule. For instance, the IBM middleware product used as one of the systems in the empirical study in this paper is a case in point. While it takes over four hours to execute the entire test suite for this system, the typical smoke test scenario performed after each code submit is assigned only an hour or less of testing time, forcing the tester to select a subset of tests from the available pool. If the computation involved in test suite minimisation requires more than one hour itself, then the tester cannot benefit from such a technique; the smoke test will be highly suboptimal as a result. Using the GPGPU approach introduced in this paper, this time was reduced from over an hour to just under 3 minutes, thereby allowing sophisticated minimisation to be used on standard machines without compromising the overall build cycle.

The paper presents a modified evolutionary algorithm for the multi-objective regression test minimisation problem. The algorithm is modified to support implementation on a GPU by transforming the fitness evaluation of the population of individual solutions into a matrix-multiplication problem, which is inherently parallel and renders itself very favourably to the GPGPU approach. This transformation to matrix-multiplication is entirely straightforward and may well be applicable to other SBSE problems, allowing them to benefit from similar scale-ups to those reported in this paper.

This algorithm has been implemented using `OpenCL` technology, a framework for GPGPU. The paper reports the results of the application of the parallelised GPGPU algorithm on 13 real world programs, including widely studied, but relatively small examples from the Siemens' suite [15], through larger more realistic real world examples from the Software-Infrastructure Repository (SIR) for testing [16], and on a very large IBM middleware regression testing problem.

The primary contributions of the paper are as follows:

1. The paper is the first to develop SBSE algorithms for GPGPU as a mechanism for affordable massive parallelism.
2. The paper presents results for real world instances of the multi objective test suite minimisation problem. The results indicate that dramatic speed–up is achievable. For the systems used in the empirical study, speed–ups over 20x were observed. The empirical evidence suggests that, for larger problems where the scale up is the most needed, the degree of speed–up is the most dramatic; a problem that takes over an hour using conventional techniques, can be solved in minutes using the GPGPU approach. This has important practical ramifications because regression testing cycles are often compressed: overnight build cycles are not uncommon.
3. The paper studies multiple evolutionary algorithms and both GPU- and CPU-based parallelisation methods in order to provide robust empirical evidence for the scalability conferred by the use of GPGPU. The GPGPU parallelisation technique maintained the same level of speed–up across all algorithms studied. The empirical evidence highlights the limitations of CPU-based parallelisation: with smaller problems, multi-threading overheads erode the speed–up, whereas with larger problems it fails to scale as well as GPU-based parallelisation.
4. The paper explores the factors that influence the degree of speed–up achieved, revealing that both program size and test suite size are closely correlated to the degree of speed–up achieved. The data have a good fit to a model for which increases in the degree of scale up achieved are logarithmic in both program and test suite size.

The rest of the paper is organised as follows. Section 2 presents background and related work in test suite minimisation and GPGPU-based evolutionary computation. Section 3 describes how the test suite minimisation problem is re-formulated for a parallel algorithm, which is described in detail in Section 4. Section 5 describes the details of the empirical study, the results of which are analysed in Section 6. Section 7 discusses the related work and Section 8 concludes.

2 Background

Multi-Objective Test Suite Minimisation: The need for test suite minimisation arises when the regression test suite of an existing software system grows to such an extent that it may no longer be feasible to execute the entire test suite [17]. In order to reduce the size of the test suite, any *redundant* test cases in the test suite need to be identified and removed. More formally, test suite minimisation problem can be defined as follows [18]:

Test Suite Minimisation Problem

Given: A test suite of m tests, T, a set of l test goals $\{r_1, \dots, r_l\}$, that must be satisfied to provide the desired 'adequate' testing of the program, and subsets of T, T_is, one associated with each of the r_is such that any one of the test cases t_j belonging to T_i can be used to achieve requirement r_i.

Problem: Find a representative set, T', of test cases from T that satisfies all r_is.

The testing criterion is satisfied when every test-case requirement in $\{r_1, \dots, r_l\}$ is satisfied. A test-case requirement, r_i, is satisfied by any test case, t_j, that belongs to T_i, a subset of T. Therefore, the representative set of test cases is the hitting set of T_is. Furthermore, in order to maximise the effect of minimisation, T' should be the minimal hitting set of T_is. The minimal hitting-set problem is an NP-complete problem as is the dual problem of the minimal set cover problem [19].

The NP-hardness of the problem encouraged the use of heuristics and meta-heuristics. The greedy approach [20] as well as other heuristics for minimal hitting set and set cover problem [21, 22] have been applied to test suite minimisation but these approaches were not cost-cognisant and only dealt with a single objective (test coverage). With the single-objective problem formulation, the solution to the test suite minimisation problem is one subset of test cases that maximises the test coverage with minimum redundancy.

Since the greedy algorithm does not cope with multiple objectives very well, Multi-Objective Evolutionary Algorithms have been applied to the multi-objective formulation of the test suite minimisation [23, 24]. While this paper studies three selected MOEAs, the principle of parallelising fitness evaluation of multiple solutions in the population of an MOEA applies universally to any MOEA.

GPGPU and Evolutionary Algorithms: Graphics cards have become a compelling platform for intensive computation, with a set of resource-hungry graphic manipulation problems that have driven the rapid advances in their performance and programmability [12]. As a result, consumer-level graphics cards boast tremendous memory bandwidth and computational power. For example, ATI Radeon HD4850 (the graphics card used in the empirical study in the paper), costing about \$150 as of April 2010, provides 1000GFlops processing rate and 63.6GB/s memory bandwidth. Graphics cards are also becoming faster more quickly compared to CPUs. In general, it has been reported that the computational capabilities of graphics cards, measured by metrics of graphics performance, have compounded at the average yearly rate of 1.7x (rendered pixels/s) to 2.3x (rendered vertices/s) [12]. This significantly outperforms the growth in traditional microprocessors; using the SPEC benchmark, the yearly rate of growth for CPU performance has been measured at 1.4x by a recent survey [25].

The disparity between two platforms is caused by the different architecture. CPUs are optimised for executing sequential code, whereas GPUs are optimised for executing the same instruction (the graphics shader) with data parallelism (different objects on the screen). This Single-Instruction/Multiple-Data (SIMD) architecture facilitates hardware-controlled massive data parallelism, which results in the higher performance.

It is precisely this massive data-parallelism of General-Purpose computing on Graphics Processing Units (GPGPU) that presents GPGPU as an ideal platform for parallel evolutionary algorithms. Many of these algorithms require the calculation of fitness (single instruction) for multiple individual solutions in the population pool (multiple data). Early work has exploited this potential for parallelism with both single- and multi-objective evolutionary algorithms [26–28]. However, most existing evaluation has been performed on benchmark problems rather than practical applications.

3 Parallel Formulation of MOEA Test Suite Minimisation

Parallel Fitness Evaluation: The paper considers, for parallelisation, a multi objective test suite minimisation problem from existing work [24]. In order to parallelise test suite minimisation, the fitness evaluation of a generation of individual solutions for the test suite minimisation problem is re-formulated as a matrix multiplication problem. Instead of computing the two objectives (i.e. coverage of test goals and execution cost) for each individual solution, the solutions in the entire population are represented as a matrix, which in turn is multiplied by another matrix that represents the trace data of the entire test suite. The result is a matrix that contains information for both test goal coverage and execution cost. While the paper considers structural coverage as test goal, the proposed approach is equally applicable to other testing criteria, such as data-flow coverage and functional coverage provided that there is a clear mapping between tests and the test objectives they achieve.

More formally, let matrix A contain the trace data that capture the test goals achieved by each test; the number of rows of A equals the number of test goals to be covered, l, and the number of columns of A equals the number of test cases in the test suite, m. Entry $a_{i,j}$ of A stores 1 if the test goal f_i was executed (i.e. covered) by test case t_j, 0 otherwise.

The multiplier matrix, B, is a representation of the current population of individual solutions that are being considered by a given MOEA. Let B be an m-by-n matrix, where n is the size of population for the given MOEA. Entry $b_{j,k}$ of B stores 1 if test case t_j is selected by the individual p_k, 0 otherwise.

The fitness evaluation of the entire generation is performed by the matrix multiplication of $C = A \times B$. Matrix C is a l-by-n matrix; entry $c_{i,k}$ of C denotes the number of times test goal f_i was covered by different test cases that had been selected by the individual p_k.

Cost and Coverage In order to incorporate the execution cost as an additional objective to the MOEA, the basic reformulation is extended with an extra row in matrix A. The new matrix, A', is an $l+1$ by m matrix that contains the cost of each individual test case in the last row. The extra row in A' results in an additional row in C' which equals to $A' \times B$ as follows:

$$A' = \begin{pmatrix} a_{1,1} & \dots & a_{1,m} \\ a_{2,1} & \dots & a_{2,m} \\ & \dots & \\ a_{l,1} & \dots & a_{l,m} \\ cost(t_1) & \dots & cost(t_m) \end{pmatrix} \qquad C' = \begin{pmatrix} c_{1,1} & \dots & c_{1,n} \\ c_{2,1} & \dots & c_{2,n} \\ & \dots & \\ c_{l,1} & \dots & c_{l,n} \\ cost(p_1) & \dots & cost(p_n) \end{pmatrix}$$

By definition, an entry $c_{l+1,k}$ in the last row in C' is defined as $c_{l+1,k} = \sum_{j=1}^{m} a_{l+1,j} \cdot b_{j,k} = \sum_{j=1}^{m} cost(t_j) \cdot b_{j,k}$. That is, $c_{l+1,k}$ equals the sum of costs of all test cases selected by individual solution p_k, i.e. $cost(p_k)$. Similarly, after the multiplication, the k-th column of matrix C' contains the coverage of test goals achieved by individual solution p_k. However, this information needs to be summarised into a percentage coverage, using a step function f as follows: $coverage(p_k) = \frac{\sum_{i=1}^{m} f(c_{i,k})}{m}$, $f(x) = 1$ $(x > 0)$ or 0 (otherwise).

While the cost objective is calculated as a part of the matrix multiplication, the coverage of test goals requires a separate step to be performed. Each column of C' contains the number of times individual testing goals were *covered* by the corresponding solution; in order to calculate the coverage metric for a solution, it is required to iterate over the corresponding column of C'. However, the coverage calculation is also of highly parallel nature because each column can be independently iterated over and, therefore, can take the advantage of GPGPU architecture by running n threads.

4 Algorithms

This section presents the parallel fitness evaluation components for CPU and GPU and introduces the MOEAs that are used in the paper.

Parallel Matrix Multiplication Algorithm: Matrix multiplication is inherently parallelisable as the calculation for an individual entry of the product matrix does not depend on the calculation of any other entry. Algorithm 1 shows the pseudo-code of the parallel matrix multiplication algorithm using the matrix notation in Section 3.

Algorithm 1 uses one thread per element of matrix C', resulting in a total of $(l+1) \cdot n$ threads. Each thread is identified with unique thread id, *tid*. Given a thread id, Algorithm 1 calculates the corresponding element of the resulting matrix, $C'_{y,x}$ given the width of matrix A, w_A, i.e., $y = \frac{tid}{w_A}$ and $x = tid \mod w_A$.

Coverage Collection Algorithm: After matrix-multiplication using Algorithm 1, coverage information is collected using a separate algorithm whose pseudo-code is shown in Algorithm 2. Unlike Algorithm 1, the coverage collection algorithm only requires n threads, i.e. one thread per column in matrix C'.

The loop in Line (3) and (4) counts the number of structural elements that have been executed by the individual solution p_{tid}. The coverage is calculated by dividing this number by the total number of structural elements that need to be covered.

While coverage information requires a separate collection phase, the sum of costs for each individual solution has been calculated by Algorithm 1 as a part of the matrix multiplication following the extension in Section 3.

Algorithm 1: Parallel Matrix Multiplication

Input: The thread id, tid, arrays containing $l+1$ by m and m by n matrices, A and B, the width of matrix A and B, w_A and w_B

Output: An array to store an $l+1$ by n matrix, C

```
MatMult($tid$, $A$, $B$, $w_A$, $w_B$)
(1)  $x \leftarrow tid \bmod w_A$
(2)  $y \leftarrow \frac{tid}{w_A}$
(3)  $v \leftarrow 0$
(4)  for $k = 0$ to $w_A - 1$
(5)      $v \leftarrow v + A[y \cdot w_A + k] \cdot B[k \cdot w_B + x]$
(6)  $C'[y * w_B + x] \leftarrow v$
```

Algorithm 2: Parallel Coverage Collection

Input: The thread id, tid, an array containing the result of matrix-multiplication, C', the width of matrix A, w_A and the height of matrix A, h_A

Output: An array containing the coverage achieved by each individual solution, $coverage$

```
CollectCoverage($tid$, $C'$, $w_A$, $h_A$)
(1)  $e \leftarrow 0$
(2)  for $k = 0$ to $w_A - 1$
(3)      if $C'[k \cdot w_A + tid] > 0$ then $e \leftarrow e + 1$
(4)  $coverage[tid] \leftarrow e / h_A$
```

5 Experimental Setup

5.1 Research Questions

This section presents the research questions studied in the paper. **RQ1** and **RQ2** concern the scalability achieved by the speed-up through the use of GPGPU.

RQ1. Speed–up: what is the speed–up factor of GPU- and CPU-based parallel versions of MOEAs over the untreated CPU-based version of the same algorithms for multi-objective test suite minimisation problem?

RQ2. Correlation: what are the factors that have the highest correlation to the speed–up achieved, and what is the correlation between these factors and the resulting speed–up?

RQ1 is answered by observing the dynamic execution time of the parallel versions of the studied algorithms as well as the untreated single-threaded algorithms. For **RQ2**, two factors constitute the size of test suite minimisation problem: the number of test cases in the test suite and the number of test goals in System Under Test (SUT) that need to be covered. The speed–up values measured for **RQ1** are statistically analysed to investigate the correlation between the speed–up and these two size factors.

RQ3. Insight: what are the realistic benefits of the scalability that is achieved by the GPGPU approach to software engineers?

RQ3 concerns the practical implications of the speed-up and the consequent scalability to the practitioners. This is answered by analysing the result of test suite minimisation obtained for a real-world testing problem.

5.2 Subjects

Table 1 shows the subject programs for the empirical study. 12 of the programs and test suites are from the Software Infrastructure Repository (SIR) [16]. In order to obtain test suites with varying sizes ranging from a few hundred to a few thousand test cases, the study includes multiple test suites for some subject programs. For `printtokens` and `schedule`, smaller test suites are coverage-adequate test suites, whereas larger test suites include all the available test cases. To avoid selection bias, four small test suites were randomly selected from the pool of available tests for each program. In the case of `space`, SIR contains multiple coverage-adequate test suites of similar sizes; fout test suites were selected randomly.

The subjects also include a large system-level test suite from IBM. For this subject, the coverage information was maintained at the function level. The test suite contains only 181 test cases, but these test cases are used to cover 61,770 functions in the system.

Each test suite has an associated execution cost dataset. For the subject programs from SIR, the execution costs were measured by observing the number of instructions required by the execution of tests. This was performed using a well-known profiling tool, `valgrind` [29], which executes the given program on a virtual processor. For `ibm`, physical wall-clock time data, measured in seconds, were provided by IBM. The entire test suite for `ibm` takes more than 4 hours to execute.

Table 1. Subject programs used for the empirical study.

Subject	Description	Program Size	Test Suite Size
`printtokens`	Lexical analyser	188	315-319[2]
			4,130
`printtokens2`	Lexical analyser	199	4,115
`schedule`	Priority scheduler	142	224-227[2]
			2,650
`schedule2`	Priority scheduler	142	2,710
`tcas`	Aircraft collision avoidance system	65	1,608
`totinfo`	Statistics computation utility	124	1,052
`replace`	Pattern matching & substitution tool	242	5,545
`space`	Array Definition Language (ADL) interpreter	3,268	154-160[3]
`flex`	Lexical analyser	3,965	103
`gzip`	Compression utility	2,007	213
`sed`	Stream text editor	1,789	370
`bash`	Unix shell	6,167	1,061
`ibm`	An IBM middleware system	61,770[1]	181

[1] For the IBM middleware system, the program size represents the number of functions that need to be covered. Others are measured in LOC.
[2] For `schedule` and `printtokens`, four randomly selected, coverage-adequate test suites were used as well as the complete test suite in SIR.
[3] For `space`, four randomly selected, coverage-adequate test suites were used.

5.3 Implementation & Hardware

Implementation: The paper uses NSGA-II implementation from the open source Java MOEA library, `jMetal` [30, 31] as the untreated version of MOEA. The GPGPU-based parallel version of NSGA-II is implemented in the OpenCL GPGPU framework using a Java wrapper called `JavaCL` [32]. The CPU-based parallel version of NSGA-II uses a parallel programming library for Java called `JOMP` [33]. `JOMP` allows parameterised configuration of the number of threads to use. In both cases, the parallelisation is only applied to the fitness evaluation step of the basic `jMetal` implementation of NSGA-II, because it is not clear whether certain steps in NSGA-II, such as sorting, may yield sufficient efficiency when performed in parallel.

NSGA-II is configured with population size of 256 following the standard recommendation to set the number of threads to multiples of 32 or 64 [34]. The stopping criterion is to reach the maximum number of fitness evaluations, which is set to 64,000, allowing 250 generations to be evaluated. Individual solutions are represented by binary strings that form columns in matrix B in Section 3. The initial population is generated by randomly setting the individual bits of these binary strings so that the initial solutions are randomly distributed in the phenotype space.

NSGA-II uses the binary tournament selection operator and the single-point crossover operator with probability of crossover set to 0.9 and the single bit-flip mutation operator with the mutation rate of $\frac{1}{n}$ where n is the length of the bit-string (i.e. the number of test goals).

Hardware: All configurations of NSGA-II have been evaluated on a machine with a quad-core Intel Core i7 CPU (2.8GHz clock speed) and 4GB memory, running Mac OS X 10.6.5 with Darwin Kernel 10.6.0 for `x86_64` architecture. The Java Virtual Machine used to execute the algorithms is Java SE Runtime with version 1.6.0_22. The GPGPU-based version of NSGA-II has been evaluated on an ATI Radeon HD4850 graphics card with 800 stream processors running at 625MHz clock speed and 512MB GDDR3 onboard memory.

5.4 Evaluation

The paper compares five different configurations of NSGA-II: the untreated configuration (hereafter refered to `CPU`), the GPGPU configuration (`GPU`) and the `JOMP`-based parallel configurations with 1, 2, and 4 threads (`JOMP1/2/4`). The configuration with one thread (`JOMP1`) is included to observe the speed-up achieved by evaluating the fitness of the entire population using matrix multiplication, instead of evaluating the solutions one by one as in the untreated version. Any speed–up achieved by `JOMP1` over `CPU` is, therefore, primarily achieved by the optimisation that removes the method invocation overheads. On the other hand, `JOMP1` does incur an additional thread management overhead.

For each subject test suite, the five configurations were executed 30 times in order to cater for the inherent randomness in dynamic execution time. The observation of algorithm execution time ($Time_{total}$) is composed of the following three parts:

- Initialisation ($Time_{init}$): the time it takes for the algorithm to initialise the test suite data in a usable form; for example, `GPU` configurations of MOEAs need to transfer the test suite data onto the graphics card.
- Fitness Evaluation ($Time_{fitness}$): the time it takes for the algorithm to evaluate the fitness values of different generations during its runtime.
- Remaining ($Time_{remaining}$): the remaining parts of the execution time, most of which is used for archive management, genetic operations, etc.

Execution time is measured using the system clock. The speed-up is calculated by dividing the amount of the time that the `CPU` configuration required by the amount of the time parallel configurations required.

6 Results

Speed–up: Table 2 contains the speed–up data in more detail, whereas the statistical analysis of the raw information can be obtained from the appendix.[3] Overall, the observed paired mean speed–up ranges from 1.43`x` to 25.09`x`. The speed–up values below 1.0 show that the overhead of thread management and the additional data structure manipulation can be detrimental for the problems of sufficiently small size. However, as the problem size grows, `JOMP1` becomes faster than `CPU` with all algorithms, indicating that the amount of reduced method call overhead eventually becomes greater that the thread management overhead. With the largest dataset, `ibm`, the `GPU` configuration of NSGA-II reduces the average execution time of `CPU`, 4,347 seconds (1 hour 12 minutes and 27 seconds), into the average of 174 seconds (2 minutes and 54 seconds). The speed–up remains consistently above 5.0`x` if the problem size is larger than that of `flex`, i.e. about 400,000 (103 tests $\times$ 3,965 test goals).

To provide more detailed analysis, the observed execution time data have been compared using The Mann-Whitney 'U' test. The Mann-Whitney 'U' test is a non-parametric statistical hypothesis test, i.e. it allows the comparison of two samples with unknown distributions. The execution time data observed with `JOMP1/2/4` and `GPU` configurations were compared to those from `CPU` configuration. The null hypothesis is that there is no difference between the parallel configurations and `CPU` configuration; the alternative hypothesis is that the execution time of the parallel configurations is smaller than that of `CPU` configuration.

Table 3 contains the resulting p-values. With `JOMP1` configuration, the alternative hypothesis is rejected for 15 cases at the confidence level of 95%, providing evidence that the parallel configurations required more time than the untreated configuration(`CPU`). With all other configurations, the null hypothesis is universally rejected for all subjects, providing strong evidence that the parallel configurations required less time than the untreated configuration(`CPU`). The particular results are naturally dependent on the choice of the graphics card that has been used for the experiment. However, these results, taken together, provide strong evidence that, for test suite minimisation problems of realistic sizes, the GPGPU approach can provide a speed–up of at least 5.0`x`. This finding answers **RQ1**.

Correlation: Regarding **RQ2**, one important factor that contributes to the level of speed–up is the speed of each individual computational unit in the graphics card. The HD4850 graphics card used in the experiment contains 800 stream processor units that are normally used for the computation of geometric shading. Each of these stream processors execute a single thread of Algorithm 1, of which

[3] The detailed statistical data can be viewed at `http://www.cs.ucl.ac.uk/staff/s.yoo/gpgpu`.

Table 2. Speed–up results for NSGA-II

Subject	S_{JOMP1}	S_{JOMP2}	S_{JOMP4}	S_{GPU}
`printtokens-1`	0.83	1.21	1.54	2.14
`printtokens-2`	0.83	1.23	1.56	2.20
`printtokens-3`	0.82	1.21	1.53	2.13
`printtokens-4`	0.84	1.22	1.54	2.19
`schedule-1`	0.97	1.22	1.40	1.56
`schedule-2`	0.96	1.22	1.41	1.46
`schedule-3`	0.96	1.22	1.39	1.45
`schedule-4`	0.95	1.20	1.37	1.43
`printtokens`	0.76	1.24	1.44	4.52
`schedule`	0.69	1.08	1.26	3.38
`printtokens2`	0.72	1.18	1.37	4.38
`schedule2`	0.71	1.09	1.27	3.09
`tcas`	0.84	1.10	1.30	1.94
`totinfo`	0.90	1.28	1.61	2.50
`flex`	1.58	2.76	4.19	6.82
`gzip`	1.19	2.15	3.31	8.00
`sed`	1.02	1.87	3.04	10.28
`space-1`	1.77	3.22	5.10	10.51
`space-2`	1.86	3.34	5.19	10.88
`space-3`	1.80	3.27	5.16	10.63
`space-4`	1.76	3.25	5.12	10.54
`replace`	0.73	1.23	1.44	5.26
`bash`	1.54	2.90	4.87	25.09
`ibm`	3.01	5.55	9.04	24.85

Table 3. Mann-Whitney U test for NSGA-II

Subject	p_{JOMP1}	p_{JOMP2}	p_{JOMP4}	p_{GPU}
`printtokens-1`	1.00e+00	1.51e-11	8.46e-18	1.51e-11
`printtokens-2`	1.00e+00	1.51e-11	8.46e-18	1.51e-11
`printtokens-3`	1.00e+00	1.51e-11	8.46e-18	8.46e-18
`printtokens-4`	1.00e+00	1.51e-11	1.51e-11	1.51e-11
`schedule-1`	1.00e+00	1.51e-11	1.51e-11	1.51e-11
`schedule-2`	1.00e+00	1.51e-11	8.46e-18	1.51e-11
`schedule-3`	1.00e+00	1.51e-11	1.51e-11	1.51e-11
`schedule-4`	1.00e+00	1.51e-11	1.51e-11	1.51e-11
`printtokens`	1.00e+00	8.46e-18	8.46e-18	8.46e-18
`schedule`	1.00e+00	1.51e-11	1.51e-11	8.46e-18
`printtokens2`	1.00e+00	1.51e-11	8.46e-18	1.51e-11
`schedule2`	1.00e+00	1.51e-11	8.46e-18	8.46e-18
`tcas`	1.00e+00	8.46e-18	8.46e-18	8.46e-18
`totinfo`	1.00e+00	1.51e-11	8.46e-18	8.46e-18
`flex`	8.46e-18	8.46e-18	1.51e-11	1.51e-11
`gzip`	1.51e-11	1.51e-11	1.51e-11	1.51e-11
`sed`	2.56e-07	8.46e-18	8.46e-18	1.51e-11
`space-1`	8.46e-18	8.46e-18	1.51e-11	1.51e-11
`space-2`	8.46e-18	8.46e-18	1.51e-11	1.51e-11
`space-3`	8.46e-18	8.46e-18	8.46e-18	1.51e-11
`space-4`	8.46e-18	8.46e-18	8.46e-18	1.51e-11
`replace`	1.00e+00	8.46e-18	1.51e-11	8.46e-18
`bash`	8.46e-18	8.46e-18	8.46e-18	8.46e-18
`ibm`	1.51e-11	8.46e-18	8.46e-18	1.51e-11

there exist more than 800. Therefore, if the individual stream processor is as powerful as a single core of the CPU, the absolute upper bound of speed–up would be 800. In practice, the individual stream processors run with the clock speed of 625MHz, which makes them much slower and, therefore, less powerful than a CPU core. This results in speed–up values lower than 800.

In order to answer **RQ2**, statistical regression analysis was performed on the correlation between the observed speed–up and the factors that characterise the size of problems.

Three size factors have been analysed for the statistical regression: the number of test goals and the number of test cases are denoted by l and m respectively, following the matrix notation in Section 3: l denotes the number of threads the GPGPU-version of the algorithm has to execute (as the size of the matrix C' is l-by-n and n is fixed); m denotes the amount of computation that needs to be performed by a single thread (as each matrix-multiplication kernel computes a loop with m iterations). In addition to these measurement, another size factor $z = l \cdot m$ is considered to represent the *perceived* size of the minimisation problem. Table 4 shows the results of Spearman's rank correlation analysis between size factors and observed speed–ups.

Spearman's rank correlation is a non-parametric measure of how well the relationship between two variables can be described using a monotonic function. As one variable increases, the other variable will tend to increase monotonically if the coefficient is close to 1, whereas it would decrease monotonically if the coefficient is close to -1.

Table 4. Spearman's rank correlation coefficients between three size factors and speed–ups

Config	ρ_z	ρ_l	ρ_m
JOMP1	0.2257	0.6399	-0.8338
JOMP2	0.4908	0.7800	-0.6423
JOMP4	0.4788	0.8227	-0.6378
GPGPU	0.8760	0.8617	-0.2299

Table 5. Regression Analysis for NSGA-II

Config	Model	α	β	γ	R^2
JOMP1	$S_p \sim z$	1.56e-07	N/A	1.00e+00	0.4894
	$S_p \sim \log z$	2.01e-01	N/A	-1.34e+00	0.3423
	$S_p \sim l + m$	3.27e-05	-1.13e-04	1.17e+00	0.7060
	$S_p \sim \log l + m$	2.69e-01	-4.83e-05	-4.79e-01	0.8487
	$S_p \sim l + \log m$	3.12e-05	-1.78e-01	2.15e+00	0.7600
	$S_p \sim \log l + \log m$	2.62e-01	-6.83e-02	-6.15e-02	0.8509
JOMP2	$S_p \sim z$	3.24e-07	N/A	1.58e+00	0.5009
	$S_p \sim \log z$	4.78e-01	N/A	-4.05e+00	0.4606
	$S_p \sim l + m$	6.64e-05	-1.82e-04	1.87e+00	0.6367
	$S_p \sim \log l + m$	6.00e-01	-2.84e-05	-1.83e+00	0.9084
	$S_p \sim l + \log m$	6.35e-05	-3.07e-01	3.58e+00	0.6836
	$S_p \sim \log l + \log m$	5.96e-01	-4.04e-02	-1.59e+00	0.9086
JOMP4	$S_p \sim z$	5.80e-07	N/A	2.15e+00	0.5045
	$S_p \sim \log z$	8.72e-01	N/A	-8.13e+00	0.4814
	$S_p \sim l + m$	1.16e-04	-3.42e-04	2.70e+00	0.6199
	$S_p \sim \log l + m$	1.08e+00	-5.93e-05	-4.00e+00	0.9322
	$S_p \sim l + \log m$	1.11e-04	-5.49e-01	5.74e+00	0.6611
	$S_p \sim \log l + \log m$	1.08e+00	-5.50e-02	-3.72e+00	0.9313
GPU	$S_p \sim z$	2.25e-06	N/A	4.13e+00	0.7261
	$S_p \sim \log z$	3.45e+00	N/A	-3.66e+01	0.7178
	$S_p \sim l + m$	3.62e-04	-1.63e-04	5.33e+00	0.4685
	$S_p \sim \log l + m$	3.53e+00	7.79e-04	-1.66e+01	0.8219
	$S_p \sim l + \log m$	3.62e-04	-1.34e-01	5.98e+00	0.4676
	$S_p \sim \log l + \log m$	3.85e+00	1.69e+00	-2.82e+01	0.8713

Size factor l shows the strongest overall positive correlation with speed–ups in all configurations. The correlation coefficients for z are weaker than those for l, whereas correlation for m remains negative for all algorithms and configurations.

To gain further insights into the correlation between size factors and speed–ups, a regression analysis was performed. Factor z is considered in isolation, whereas l and m are considered together; each variable has been considered in its linear form (z, l and m) and logarithmic form ($\log z, \log l$ and $\log m$). This results in 6 different combinations of regression models. Table 5 presents the results of regression analysis for four configurations respectively.

With a few exceptions of very small margins (`JOMP4`), the model with the highest r^2 correlation for all configurations is $S_p = \alpha \log l + \beta \log m + \gamma$. Figure 1 shows the 3D plot of this model for the GPU and `JOMP4` configurations.

The observed trend is that the inclusion of $\log l$ results in higher correlation, whereas models that use l in its linear form tend to result in lowest correlation. This confirms the results of Spearman's rank correlation analysis in Table 4. The coefficients for the best-fit regression model for `GPU`, $S_p = \alpha \log l + \beta \log m + \gamma$, can explain why the speed–up results for `space` test suites are higher than those for test suites with z values such as `tcas`, `gzip` and `replace`. Apart from `bash` and `ibm`, `space` has the largest number of test goals to cover, i.e. l. Since α is more than twice larger than β, a higher value of l has more impact to S_p than m.

Based on the analysis, **RQ2** is answered as follows: the observed speed–up shows a strong linear correlation to the log of the number of test goals to cover and the log of the number of test cases in the test suite. The positive correlation provides a strong evidence that GPU-based parallelisation scales up.

Furthermore, within the observed data, the speed–up continues to increase as the problem size grows, which suggests that the graphics card did not reach its full computational capacity. It may be that for larger problems, if studied, the speed–up would be even greater than those observed in this paper; certainly the correlation observed indicates that this can be expected. The finding that the scalability factor increases with overall problem size is a very encouraging finding; as the problem gets harder, the solutions are obtained faster.

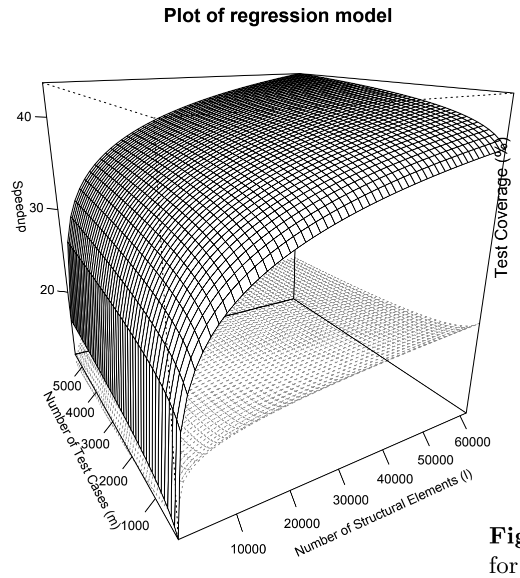

Fig. 1. 3D-plot of regression model $S_p = \alpha \log l + \beta \log m + \gamma$ for GPU(solid line) and JOMP4(dotted line) configurations.

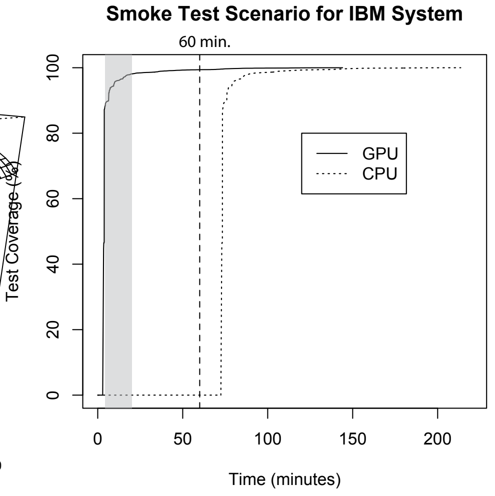

Fig. 2. Comparison of smoke test scenarios for IBM System (`ibm`). The solid line shows the trade-offs between time and test coverage when `GPU` configuration of NSGA-II is used, whereas the dotted line shows that of `CPU`. The grey area shows the interesting trade-off that the `CPU` configuration fails to exploit within 60 minutes.

Insights: This section discusses a possible real-world scenario in which the parallelisation of multi-objective test suite minimisation can have a high impact. A smoke test is a testing activity that is usually performed in a very short window of time to detect the most obvious faults, such as system crashes. IBM's smoke test practice is to allow from 30 to 60 minutes of time to execute a subset of tests from a large test suite that would require more than 4 hours to execute in its entirity.

Figure 2 shows two possible smoke test scenarios based on the results of `CPU` and `GPU` configurations of NSGA-II. The solid line represents the scenario based on the `GPU` configuration of the algorithm, whereas the dotted line represents the scenario based on the `CPU` configuration. The flat segment shows the time each configuration spends on the optimisation process; the curved segment shows the trade-off between time and test coverage achieved by the optimised test suite. Since the `CPU` configuration of NSGA-II takes longer than 60 minutes to terminate, it cannot contribute to any smoke test scenario that must be completed within 60 minutes. On the other hand, the `GPU` configuration allows the tester to consider a subset of tests that can be executed under 30 minutes. If the grey region was wider than Figure 2, the difference between two configurations would have been even more dramatic.

This answers **RQ3** as follows: a faster execution of optimisation algorithms enables the tester not only to use the algorithms but also to exploit their results more effectively. This real world smoke test example from IBM demonstrates that scale–ups accrued from the use of GPU are not only sources of efficiency improvement, they can also make possible test activities that are simply impossible without this scalability.

The ability to execute a sophisticated optimisation algorithm within a relatively short time also allows the tester to consider state-of-the-art regression testing techniques with greater flexibility. The greater flexibility is obtained because the cost of the optimisation does not have to be amortised across multiple iterations. Many state-of-the-art regression testing techniques require the use of continuously changing sets of testing data, such as recent fault history [24] or the last time a specific test case has been executed [35,36]. In addition to the use of dynamic testing data, the previous work also showed that repeatedly using the same subset of a large test suite may impair the fault detection capability of the regression testing [37].

7 Related Work

Test suite minimisation aims to reduce the number of tests to be executed by calculating the minimum set of tests that are required to satisfy the given test requirements. The problem has been formulated as the minimal hitting set problem [21], which is NP-hard [19].

Various heuristics for the minimal hitting set problem, or the minimal set cover problem (the duality of the former), have been suggested for the test suite minimisation [20, 38]. However, empirical evaluations of these techniques have reported conflicting views on the impact on fault detection capability: some reported no impact [39, 40] while others reported compromised fault detection capability [17, 41].

One potential reason why test suite minimisation has negative impact on the fault detection capability is the fact that the criterion for minimisation is structural coverage; achieving coverage alone may not be sufficient for revealing faults. This paper uses the multi-objective approach based on Multi-Objective Evolutionary Algorithm (MOEA) introduced by Yoo and Harman [24]; the paper also presents the first attempt to parallelise test suite minimisation with sophisticated criteria for scalability.

Population-based evolutionary algorithms are ideal candidates for parallelisation on graphics cards [12] and existing work has shown successful implementations for classical problems. Tsutsui and Fujimoto implemented a single-objective parallel Genetic Algorithm (GA) using GPU for the Quadratic Assignment Problem (QAP) [26]. Wilson and Banzaf implemented a linear Genetic Programming (GP) algorithm on XBox360 game consoles [27]. Langdon and Banzaf implemented GP for GPU using an SIMD interpreter for fitness evaluation [11]. Wong implemented an MOEA on GPU and evaluated the implementation using a suite of benchmark problems [28]. Wong's implementation parallelised not only the fitness evaluation step but also the parent selection, crossover & mutation operator as well as the dominance checking.

Despite the highly parallelisable nature of many techniques used in SBSE, few parallel algorithms have been used. Mitchell et al. used a distributed architecture for their clustering tool `Bunch` [7]. Asadi et al. also used a distributed Server-Client architecture for Concept Location problem [9]. However, both approaches use a distributed architecture that requires multiple machines; this paper is the first work on SBSE that presents highly affordable parallelism based on GPGPU.

8 Conclusion

This paper presents the first use of GPGPU-based massive parallelism for improving scalability of regression testing, based on Search-Based Software Engineering (SBSE). The advances in GPGPU architecture and the consequent availability of parallelism provides an ideal platform for improving SBSE scalability.

The paper presents an evaluation of the GPGPU-based test suite minimisation for real-world examples that include an industry-scale test suite. The results show that the GPGPU-based optimisation can achieve a speed–up of up to 25.09x compared to a single-threaded version of the same algorithm executed on a CPU. The highest speed–up achieved by the CPU-based parallel optimisation was 9.04x. Statistical analysis shows that the speed–up correlates to the logarithmic of the problem size, i.e. the size of the program under test and the size of the test suite. This finding indicates that as the problem becomes larger, the scalability of the proposed approach increases; a very attractive finding.

References

1. I. Sommerville, *Software Engineering*, 6th ed. Addison-Wesley, 2001.
2. R. Pressman, *Software Engineering: A Practitioner's Approach*, 3rd ed. Maidenhead, Berkshire, UK.: McGraw-Hill Book Company Europe, 1992, european adaptation (1994). Adapted by Darrel Ince.
3. J. R. Cordy, "Comprehending reality - practical barriers to industrial adoption of software maintenance automation," in *IEEE International Workshop on Program Comprehension (IWPC '03)*. IEEE Computer Society, 2003, pp. 196–206.
4. P. Y. K. Chau and K. Y. Tam, "Factors affecting the adoption of open systems: An exploratory study," *MIS Quarterly*, vol. 21, no. 1, 1997.
5. G. Premkumar and M. Potter, "Adoption of computer aided software engineering (CASE) technology: An innovation adoption perspective," *Database*, vol. 26, no. 2&3, pp. 105–124, 1995.

6. E. Cantú-Paz and D. E. Goldberg, "Efficient parallel genetic algorithms: theory and practice," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2–4, pp. 221 – 238, 2000.
7. B. S. Mitchell, M. Traverso, and S. Mancoridis, "An architecture for distributing the computation of software clustering algorithms," in *IEEE/IFIP Proceedings of the Working Conference on Software Architecture (WICSA '01)*. Amsterdam, Netherlands: IEEE Computer Society, 2001, pp. 181–190.
8. K. Mahdavi, M. Harman, and R. M. Hierons, "A multiple hill climbing approach to software module clustering," in *IEEE International Conference on Software Maintenance*. Los Alamitos, California, USA: IEEE Computer Society Press, Sep. 2003, pp. 315–324.
9. F. Asadi, G. Antoniol, and Y. Guéhéneuc, "Concept locations with genetic algorithms: A comparison of four distributed architectures," in *Proceedings of* 2^{nd} *International Symposium on Search based Software Engineering (SSBSE 2010)*. Benevento, Italy: IEEE Computer Society Press, 2010, p. To Appear.
10. Y. Zhang, "SBSE repository," www.sebase.org/sbse/publications/repository.html. Accessed February 14^{th} 2011.
11. W. B. Langdon and W. Banzhaf, "A SIMD interpreter for genetic programming on GPU graphics cards," in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, ser. Lecture Notes in Computer Science, M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, and E. Tarantino, Eds., vol. 4971. Springer, March 2008, pp. 73–85.
12. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.
13. M. Boyer, D. Tarjan, S. T. Acton, and K. Skadron, "Accelerating leukocyte tracking using cuda: A case study in leveraging manycore coprocessors," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2009.
14. N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha, "Gputerasort: High performance graphics coprocessor sorting for large database management," in *ACM SIGMOD*, 2006.
15. M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proceedings of the 16th International Conference on Software Engineering (ICSE 1994)*. IEEE Computer Society Press, May 1994, pp. 191–200.
16. H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact." *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.
17. G. Rothermel, M. Harrold, J. Ronne, and C. Hong, "Empirical studies of test suite reduction," *Software Testing, Verification, and Reliability*, vol. 4, no. 2, pp. 219–249, December 2002.
18. S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation: A survey," *Software Testing, Verification, and Reliability*, vol. *to appear*, 2010.
19. M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*. New York, NY: W. H. Freeman and Company, 1979.
20. J. Offutt, J. Pan, and J. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proceedings of the 12th International Conference on Testing Computer Software*. ACM Press, June 1995, pp. 111–123.

21. M. J. Harrold, R. Gupta, and M. L. Soffa, “A methodology for controlling the size of a test suite,” *ACM Transactions on Software Engineering and Methodology*, vol. 2, no. 3, pp. 270–285, 1993.
22. T. Chen and M. Lau, “Heuristics towards the optimization of the size of a test suite,” in *Proceedings of the 3rd International Conference on Software Quality Management*, vol. 2, 1995, pp. 415–424.
23. C. L. B. Maia, R. A. F. do Carmo, F. G. de Freitas, G. A. L. de Campos, and J. T. de Souza, “A multi-objective approach for the regression test case selection problem,” in *Proceedings of Anais do XLI Simpòsio Brasileiro de Pesquisa Operacional (SBPO 2009)*, 2009, pp. 1824–1835.
24. S. Yoo and M. Harman, “Pareto efficient multi-objective test case selection,” in *Proceedings of International Symposium on Software Testing and Analysis*. ACM Press, July 2007, pp. 140–150.
25. M. Ekman, F. Warg, and J. Nilsson, “An in-depth look at computer performance growth,” *SIGARCH Computer Architecture News*, vol. 33, no. 1, pp. 144–147, 2005.
26. S. Tsutsui and N. Fujimoto, “Solving quadratic assignment problems by genetic algorithms with GPU computation: a case study,” in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO 2009)*. ACM Press, July 2009, pp. 2523–2530.
27. G. Wilson and W. Banzhaf, “Deployment of cpu and gpu-based genetic programming on heterogeneous devices,” in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO 2009)*. New York, NY, USA: ACM Press, July 2009, pp. 2531–2538.
28. 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 (GECCO 2009)*. New York, NY, USA: ACM Press, July 2009, pp. 2515–2522.
29. N. Nethercote and J. Seward, “Valgrind: A program supervision framework,” in *Proceedings of ACM Conference on Programming Language Design and Implementation*. ACM Press, June 2007, pp. 89–100.
30. J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, “jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics,” Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, Tech. Rep. ITI-2006-10, December 2006.
31. J. Durillo, A. Nebro, and E. Alba, “The jmetal framework for multi-objective optimization: Design and architecture,” in *Proceedings of Congress on Evolutionary Computation 2010*, Barcelona, Spain, July 2010, pp. 4138–4325.
32. O. Chafik, “JavaCL: opensource Java wrapper for OpenCL library,” 2009, code.google.com/p/javacl/. Accessed June 6th 2010.
33. J. M. Bull, M. D. Westhead, M. E. Kambites, and J. Obrzalek, “Towards openmp for java,” in *Proceedings of the 2nd European Workshop on OpenMP*, 2000, pp. 98–105.
34. *ATI Stream Computing: OpenCL Programming Guide Rev. 1.05*. AMD Corp., August 2010.
35. J.-M. Kim and A. Porter, “A history-based test prioritization technique for regression testing in resource constrained environments,” in *Proceedings of the 24th International Conference on Software Engineering*. ACM Press, May 2002, pp. 119–129.
36. E. Engström, P. Runeson, and G. Wikstrand, “An empirical evaluation of regression testing based on fix-cache recommendations,” in *Proceedings of the 3rd Inter-*

national Conference on Software Testing Verification and Validation (ICST 2010). IEEE Computer Society Press, = 2010, pp. 75–78.

37. S. Yoo, M. Harman, and S. Ur, “Measuring and improving latency to avoid test suite wear out,” in *Proceedings of the Interntional Conference on Software Testing, Verification and Validation Workshop (ICSTW 2009)*. IEEE Computer Society Press, April 2009, pp. 101–110.
38. T. Y. Chen and M. F. Lau, “Dividing strategies for the optimization of a test suite,” *Information Processing Letters*, vol. 60, no. 3, pp. 135–141, 1996.
39. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, “Effect of test set minimization on fault detection effectiveness,” *Software Practice and Experience*, vol. 28, no. 4, pp. 347–369, April 1998.
40. W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, “Test set size minimization and fault detection effectiveness: A case study in a space application,” *The Journal of Systems and Software*, vol. 48, no. 2, pp. 79–89, October 1999.
41. G. Rothermel, S. Elbaum, A. Malishevsky, P. Kallakuri, and B. Davia, “The impact of test suite granularity on the cost-effectiveness of regression testing,” in *Proceedings of the 24th International Conference on Software Engineering (ICSE 2002)*. ACM Press, May 2002, pp. 130–140.