GPGPU test suite minimisation: search based software engineering performance improvement using graphics cards

Shin Yoo · Mark Harman · Shmuel Ur

Abstract It has often been claimed that SBSE uses so-called 'embarrassingly parallel' algorithms that will imbue SBSE applications with easy routes to dramatic performance improvements. However, despite recent advances in multicore computation, this claim remains largely theoretical; there are few reports of performance improvements using multicore SBSE. This paper shows how inexpensive General Purpose computing on Graphical Processing Units (GPGPU) can be used to massively parallelise suitably adapted SBSE algorithms, thereby making progress towards cheap, easy and useful SBSE parallelism. The paper presents results for three different algorithms: NSGA2, SPEA2, and the Two Archive Evolutionary Algorithm, all three of which are adapted for multi-objective regression test selection and minimization. The results show that all three algorithms achieved performance improvements up to 25 times, using widely available standard GPUs. We also found that the speed-up was observed to be statistically strongly correlated to the size of the problem instance; as the problem gets harder the performance improvements also get better.

Keywords Search based software engineering · GPGPU · Test suite minimisation · Regression testing

S. Yoo (✉) · M. Harman
University College London, Gower Street, London WC1E 6BT, UK
e-mail: shin.yoo@ucl.ac.uk

M. Harman
e-mail: mark.harman@ucl.ac.uk

S. Ur
University of Bristol, Senate House, Tyndall Avenue, Bristol BS8 1TH, UK
e-mail: shmuel.ur@gmail.com

Ⓐ Springer

1 Introduction

Search Based Software Engineering (SBSE) is a promising sub-area within Software Engineering that reformulates Software Engineering problems as search-based optimisation problems (Clark et al. 2003; Harman 2007; Harman and Jones 2001). There has been much recent interest in SBSE, with over 800 papers on the topic and several recent surveys (Ali et al. 2010; Afzal et al. 2009; Räihä 2009; Harman et al. 2012).

One weakness shared by many SBSE techniques is their significant execution time. It is an inherent drawback because many meta-heuristic optimisation algorithms are designed in such a way that they evaluate a large number of potential candidates to arrive at the set of proposed solutions. This process is often computationally demanding and can render an SBSE technique infeasible in practice, because it would simply require too much time to optimise for complex real-world SE problems. Lack of scalability has been shown to be an important barrier to wider uptake of Software Engineering research (Cordy 2003; Chau and Tam 1997; Premkumar and Potter 1995).

Fortunately, many of the algorithms used in SBSE, such as the most widely used evolutionary algorithms (Harman et al. 2012), are classified as 'embarrassingly parallel' due to their inherent potential for parallelism. The high computational requirement created by the necessity to examine many solutions does not necessitate long elapsed time as the examinations can be done in parallel. The computation process (fitness evaluation) for each candidate solution is identical, thereby making the overall process well-suited to Single Instruction Multiple Data (SIMD) parallelism.

The use of multicore computing is rapidly becoming commonplace, with very widely available and inexpensive platforms that offer several hundreds of processing elements that implement SIMD parallelism. Furthermore, it is likely that we shall see significant advances in such platforms in the near future, with thousands and perhaps tens of thousands of simple processing elements becoming available within the reach of 'standard' desktop computation.

Many SBSE fitness functions are ideally-suited to such simple processing elements. However, there has been little work on multicore SBSE. The first authors to suggest the use of multicore SBSE were Mitchell et al. (2001) who used a distributed architecture to parallelise modularisation through the application of search-based clustering. Subsequently, Mahdavi et al. (2003) used a cluster of standard PCs to implement a parallel hill climbing algorithm. More recently, Asadi et al. (2010) used a distributed architecture to parallelise a genetic algorithm for the concept location problem. However, hitherto, no authors[1] have used General Purpose computing on Graphical Processing Units (GPGPU) for SBSE.

In this paper we propose GPGPU SBSE; the use of GPGPU devices to achieve multicore execution of SBSE, using simple fitness computations mapped across the multiple processing elements of standard GPGPU architectures. We report results for the application of GPGPU SBSE to the multi-objective regression test selection problem (also known as test case minimization).

---

[1]This paper is an extended version of our SSBSE 2011 paper (Yoo et al. 2011a), which was the first to propose GPGPU SBSE.

In order to apply GPGPU to Software Engineering optimisation problems, the specialized graphics manipulation hardware needs to be harnessed for a purpose for which it was not originally designed. Fortunately, the recent trend towards 'general purpose' graphics processing units lends itself well to this task. The underlying firmware typically implements certain matrix manipulation operations very efficiently. The key to unlocking the potential of GPGPU therefore lies in the ability to reformulate the optimisation problem in terms of these specific matrix manipulation operations.

This paper focusses on the problem of Search Based Regression Testing, which is one problem in the general area of Search Based Software Testing. Regression Testing is concerned with the process of re–testing software after change. After each change to the system, the pool of available test data needs to be re-executed in order to check whether change has introduced new faults. Regression Testing therefore seeks to answer the question 'has the software regressed?'. There have been several survey papers on Regression Testing applications and techniques that provide a more detailed treatment (Harrold and Orso 2008; Yoo and Harman 2012a; Engström et al. 2009).

In search based regression testing, the goal is to use search based optimisation algorithms to find optimal sets of test cases (regression test suite minimisation Yoo and Harman 2007) or to order test cases for regression testing (regression test prioritisation Walcott et al. 2006; Li et al. 2007). This paper concentrates upon the former problem of regression test minimisation. Recent results have shown that this is a promising area of SBSE application; the results obtained from the SBSE algorithms have been shown to be human competitive (de Souza et al. 2010).

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. A multi-objective approach to test suite minimisation (Yoo and Harman 2007) provides an ideal solution as it can recommend subsets of tests that can be executed within different time budgets. However, as the selection of tests in the smoke tests is not static and depends on the code submitted, the given time budget should account for both the computation involved in test suite minimisation and for running the tests. Therefore it is important that test suite optimization will be done in a small fraction of the time, thereby allowing sophisticated minimisation to be used on standard machines.

The paper modifies three widely used evolutionary algorithms (SPEA2, NSGA2 and the Two Archive Algorithm) for the multi-objective regression test minimisation problem. The algorithms are 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.

The modified algorithms have been implemented using `OpenCL` technology, a framework for GPGPU. The paper reports the results of the application of the

parallelised GPGPU algorithms on 13 real-world programs, including widely studied, but relatively small examples from the Siemens' suite (Hutchins et al. 1994), through larger more realistic real-world examples from the Software-Infrastructure Repository (SIR) for testing (Do et al. 2005), and on a very large IBM middleware regression testing problem.

The primary contributions of the paper are as follows:

1. 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 25× 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.

2. The paper studies three different multi-objective evolutionary algorithms based on both GPU- and CPU-based parallelisation methods 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.

3. 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 material on 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 threats to validity and Section 8 presents the related work. Section 9 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 (Rothermel et al. 2002b). In order to reduce the size of the test suite, any *redundant* test cases in the test suite need to be identified and removed. One widely accepted criterion for redundancy is defined in relation to the coverage achieved by test cases (Rothermel et al. 1998; Black et al. 2004). If the test coverage achieved by test case t_1 is a subset of the test coverage achieved by test case t_2, it can be said that the execution of t_1

is redundant as long as t_2 is also executed. The aim of test suite minimisation is to obtain the smallest subset of test cases that are not redundant with respect to a set of test requirements. More formally, test suite minimisation problem can be defined as follows (Yoo and Harman 2012a):

2.1 Test Suite Minimisation Problem

Given A test suite of m tests, T, a set of l test requirements $\{r_1, \ldots, r_l\}$, 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 minimal representative set,[2] T', of test cases from T that satisfies all r_is.

The testing criterion is satisfied when every test-case requirement in $\{r_1, \ldots, r_l\}$ 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. If we represent test cases as vertices of a bipartite graph on the left side, and the requirements on the right side, and the satisfiability relationship as edges between two sides, the minimal representative set of test cases is the hitting set of T_is (i.e. the subset of vertices on the left, the union of whose connected right side vertices equals the set of all requirements). Furthermore, in order to maximise the effect of minimisation, T' should be the minimal hitting set of T_is. The minimal representative set problem is an NP-complete problem as is the dual problem of the minimal set cover problem (Garey and Johnson 1979).

The NP-hardness of the problem encouraged the use of heuristics and meta-heuristics. The greedy approach (Offutt et al. 1995) as well as other heuristics for minimal hitting set and set cover problem (Harrold et al. 1993; Chen and Lau 1995) 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.

Later, the problem was reformulated as a multi-objective optimisation problem (Yoo and Harman 2007). With the multi-objective problem formulation, the solution to the test suite minimisation problem is not just a single solution but a set of non-dominated, Pareto-efficient solutions. This set of solutions reveals the trade-off between test coverage and the cost of testing that is specific to the test suite in consideration. For example, with the solution to the multi-objective test suite minimisation problem, it is possible not only to know what the minimal subset that achieves the maximum test coverage is, but also to know how much test coverage is possible for any given testing budget.

Since the greedy algorithm may not always produce Pareto optimal solutions for multi-objective test suite minimisation problems, Multi-Objective Evolutionary Algorithms (MOEAs) have been applied (Maia et al. 2009; Yoo and Harman 2007). While this paper studies three selected MOEAs, the principle of parallelising fitness

---

[2]Given a universe (in our context, all test requirements), a representative set is the set of subsets of universe (in our context, subsets of test requirements achieved by different tests) that whose union is equal to the universe.

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 (Owens et al. 2007). 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 63.6 GB/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 $1.7\times$ (rendered pixels/s) to $2.3\times$ (rendered vertices/s) (Owens et al. 2007). This significantly outperforms the growth in traditional microprocessors; using the SPEC benchmark, the yearly rate of growth for CPU performance has been measured at $1.4\times$ by a recent survey (Ekman et al. 2005).

The disparity between the 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 for certain types of problems in which a large dataset has to be submitted to the same operations.

It is precisely this massive data-parallelism of General-Purpose computing on Graphics Processing Units (GPGPU) that makes 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 (Tsutsui and Fujimoto 2009; Wilson and Banzhaf 2009; Wong 2009). However, most existing evaluation has been performed on benchmark problems rather than practical applications.

3 Parallel Formulation of MOEA for Test Suite Minimisation

Parallel Fitness Evaluation   The paper considers, for parallelisation, a multi-objective test suite minimisation problem from existing work (Yoo and Harman 2007). 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 requirements 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 the test goal, the proposed approach is equally applicable to any other testing criteria, either coverage generated such as data-flow coverage and functional coverage or even

those generated manually, provided that there is a clear mapping between tests and the test requirements they achieve.

More formally, let matrix A contain the trace data that capture the test requirements achieved by each test; the number of rows of A equals the number of test requirements 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.

$$A = \begin{pmatrix} a_{1,1} & \ldots & a_{1,m} \\ a_{2,1} & \ldots & a_{2,m} \\ & \ldots & \\ a_{l,1} & \ldots & a_{l,m} \end{pmatrix}$$

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. In other words, each column in matrix B corresponds to a vector of decision variables that denote the selected test cases.

$$B = \begin{pmatrix} b_{1,1} & \ldots & b_{1,n} \\ b_{2,1} & \ldots & b_{2,n} \\ & \ldots & \\ b_{m,1} & \ldots & b_{m,n} \end{pmatrix}$$

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:

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

By definition, an entry $c_{l+1,k}$ in the last row in C' is defined as $c_{l+1,k} = \sum_{j=1}^{m} a_{l+1,j} \cdot b_{j,k} = \sum_{j=1}^{m} \text{cost}(t_j) \cdot b_{j,k}$. That is, $c_{l+1,k}$ equals the sum of costs of all test cases *selected* ($b_{j,k}$ equals 1) by the k-th individual solution p_k, i.e. $\text{cost}(p_k)$. Similarly, after the multiplication, the k-th column of matrix C' contains the coverage of test requirements achieved by individual solution p_k. However, this information needs to be summarised into a percentage coverage, using a step function f as follows: $\text{coverage}(p_k) = \frac{\sum_{i=1}^{m} f(c_{i,k})}{l}$, $f(x) = 1$ ($x > 0$) or 0 (otherwise). The role of the step function is to translate the linear sum of how many times a test goal has been covered into boolean coverage of whether it was covered or not.

The cost objective can be calculated as a part of the matrix multiplication because it can be linearly computed from the decision variable vectors (columns of matrix B). Other objectives that share the linearity may also be computed using matrix multiplication. However, the coverage of test requirements requires a separate step to be performed. Each column of C' contains the number of times individual testing requirements 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'. The coverage calculation is highly parallel in nature because each column can be independently iterated over and, therefore, can take the advantage of GPGPU architecture by running multiple 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. $y = \frac{tid}{w_B}$ and $x = tid \mod w_B$.

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 of length $(l + 1)n$ to store matrix C'
MATMULT(*tid*, A', B, $w_{A'}$, w_B)

(1) $x \leftarrow tid \mod w_B$
(2) $y \leftarrow \lfloor tid/w_B \rfloor$
(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) $C'[y * w_B + x] \leftarrow v$

Coverage Collection Algorithm After matrix-multiplication using Algorithm 1, coverage information is collected using a separate algorithm, pseudo-code of which 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 (2) and (3) 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 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** $h_{A'} - 1$
(3)   **if** $C'[k \cdot w_{B'} + tid] > 0$ **then** $e \leftarrow e + 1$
(4) $coverage[tid] \leftarrow e/(h_{A'} - 1)$

---

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, whereas **RQ3** concerns the practical implications of the speed-up and the consequent scalability to the practitioners.

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 of the problem instances that have the highest correlation to the speed-up achieved, and what is the correlation between these factors and the resulting speed-up?

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

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 requirements 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** is answered by analysing the result of test suite minimisation obtained for a real-world testing problem.

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
schedule	Priority scheduler	142	224–227[2]
			2,650
printtokens2	Lexical analyser	199	4,115
schedule2	Priority scheduler	142	2,710
tcas	Aircraft collision avoidance system	65	1,608
totinfo	Statistics computation utility	124	1,052
flex	Lexical analyser	3,965	103
gzip	Compression utility	2,007	213
sed	Stream text editor	1,789	370
space	Array Definition Language (ADL) interpreter	3,268	154–160[3]
replace	Pattern matching & substitution tool	242	5,545
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. The coverage objective for the IBM system also denotes function coverage. For all other subject, the size and the coverage objective are measured and calculated using LOC

[2]For schedule and printtokens, four coverage-adequate test suites were randomly selected from those provided by SIR, as well as the complete test suite

[3]For space, four randomly selected, coverage-adequate test suites were used

5.2 Subjects

Table 1 shows the subject programs for the empirical study. Twelve of the programs and test suites are from the Software Infrastructure Repository (SIR) (Do et al. 2005). 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 smaller 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; four 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 (Nethercote and Seward 2007), 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 h to execute.

5.3 Implementation and Hardware

Implementation The paper uses the open source Java MOEA library, jMetal (Durillo et al. 2006, 2010) as a library of untreated versions of MOEAs: NSGA-II

and SPEA2 are included in the `jMetal` library; The Two Archive algorithm has been implemented using the infrastructure provided by the library. The untreated versions of MOEAs evaluate the fitness of individual solutions in the population one at a time, which incurs method invocations regarding the retrieval of coverage and cost information.

The GPGPU-based parallel versions of these three algorithms are implemented in the OpenCL GPGPU framework using a Java wrapper called `JavaCL` (Chafik 2009). The CPU-based parallel versions of the three algorithms use a parallel programming library for Java called `JOMP` (Bull et al. 2000). `JOMP` allows parameterised configuration of the number of threads to use. The parallelisation is only applied to the fitness evaluation step because it is not clear whether certain steps in the studied algorithms, such as sorting, may yield sufficient efficiency when performed in parallel.

All three algorithms are configured with population size of 256 following the standard recommendation to set the number of threads to multiples of 32 or 64 (ATI Stream Computing 1010). The archive size for SPEA2 and The Two Archive algorithm is also set to 256. The stopping criterion for all three algorithms is to reach the maximum number of fitness evaluations, which is set to 64,000, allowing 250 generations to be evaluated.

All three algorithms solve the test suite minimisation problem by selecting Pareto-optimal subsets of test cases, 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 and SPEA2 use the binary tournament selection operator. The Two Archive algorithm uses the uniform selection operator as described in the original paper (Praditwong and Yao 2006): the selection operator first selects one of the two archives with equal probability and then selects one solution from the chosen archive with uniform probability distribution. All three algorithms use 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 requirements).

Hardware All algorithms have been evaluated on a machine with a quad-core Intel Core i7 CPU (2.8 GHz clock speed) and 4 GB 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 versions of MOEAs have been evaluated on an ATI Radeon HD4850 graphics card with 800 stream processors running at 625 MHz clock speed and 512 MB GDDR3 onboard memory.

5.4 Evaluation

The paper compares three MOEAs, each with five different configurations: 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 versions of

MOEA. Any speed-up achieved by `JOMP1` over `CPU` is, therefore, primarily achieved by the code-level optimisation that removes the method invocation overheads. On the other hand, `JOMP1` does incur an additional thread management overhead.

In total, there are 15 different configurations (three algorithms with five configurations each). For each subject test suite, the 15 configurations were executed 30 times in order to cater for the inherent randomness in dynamic execution time, measured using the system clock. The speed-up is calculated by dividing the amount of the time that the `CPU` configuration required with the amount of the time that parallel configurations required. While we discuss the speed-up values only using the total execution time, we also provide observations of the three parts break-down of the total execution time ($Time_{\text{total}}$) for each algorithm as below:

– Initialisation ($Time_{\text{init}}$): 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 ($Time_{\text{fitness}}$): the time it takes for the algorithm to evaluate the fitness values of different generations during its runtime.
– Remaining ($Time_{\text{remaining}}$): the remaining parts of the execution time, most of which is used for archive management, genetic operations, etc.

6 Results

This section presents the speed-up measurements of the single-threaded, CPU-based multi-threaded, and GPGPU-based multi-threaded approaches and analyses the correlation between the speed-up and problem size.

6.1 Speed-up

Figure 1 presents the mean paired speed-up results of all configurations. The mean paired speed-up values were calculated by dividing the execution time of `CPU` with the corresponding execution time of the parallel configurations for each of the 30 observations. Tables 2, 3 and 4 contain the speed-up data in more detail, whereas the statistical analysis of the raw information can be obtained from Tables 12, 13 and 14 in the Appendix.

Overall, the observed paired mean speed-up ranges from 0.47× to 25.09×. While the different archive management strategies used by each MOEAs make it difficult to compare the execution time results directly (because the different amount of heap used by each may affect JVM's performance differently), it is possible to observe the general trend that the speed-up tends to increase as the problem size grows. The speed-up values below 1.0 show that the overhead of thread management and the additional communication 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.
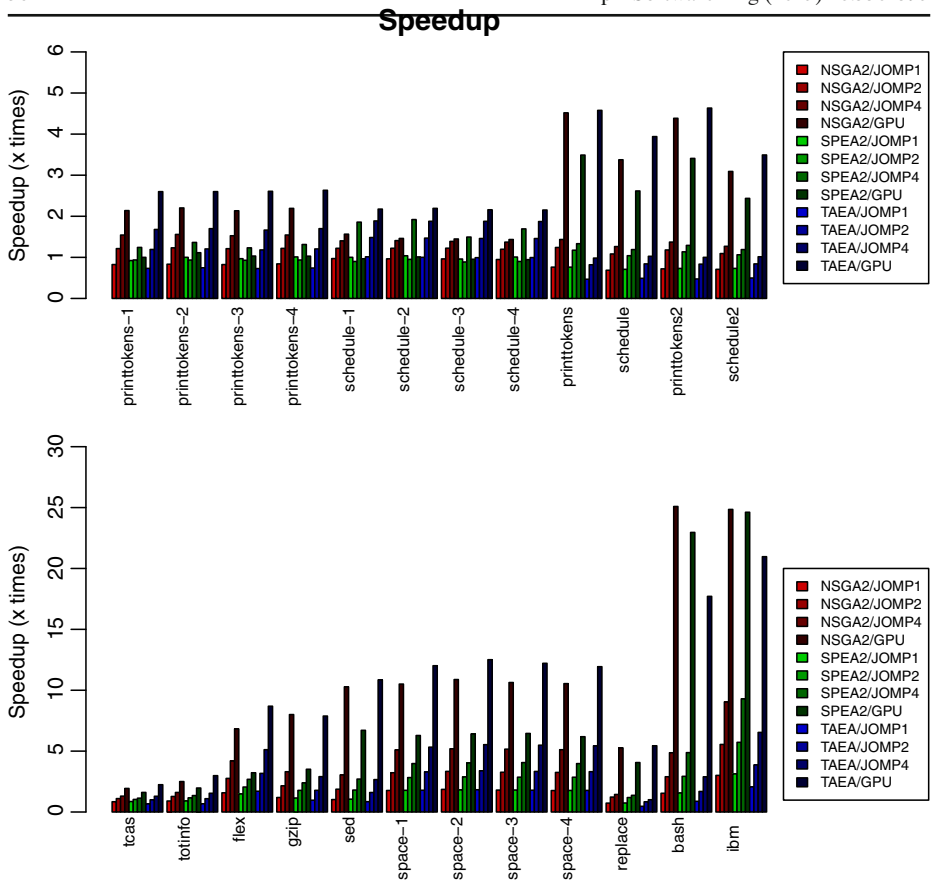
Fig. 1 Mean paired speed-ups achieved by different algorithms and parallel configurations

With the largest dataset, `ibm`, the `GPU` configuration of NSGA-II reduces the average execution time of `CPU`, 4,347 s (1 h 12 min and 27 s), to the average of 174 s (2 min and 54 s). The speed-up remains consistently above $3.0\times$ for all three algorithms if the problem size is larger than that of `flex`, i.e. about 400,000 (103 tests × 3,965 test requirements).

To provide inferential statistical analysis of 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 distribution. 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.

Tables 5, 6 and 7 present the resulting *p*-values. With `JOMP1` and `JOMP2` configurations, the alternative hypothesis is rejected for 39 and 12 cases at the

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 Speed-up results for SPEA2

Subject	S_{JOMP1}	S_{JOMP2}	S_{JOMP4}	S_{GPU}
printtokens-1	0.92	0.94	1.24	1.00
printtokens-2	1.00	0.93	1.36	1.11
printtokens-3	0.97	0.93	1.23	1.03
printtokens-4	1.01	0.94	1.31	1.03
schedule-1	1.00	0.90	1.86	0.97
schedule-2	1.04	0.95	1.92	1.01
schedule-3	0.96	0.89	1.49	0.95
schedule-4	1.01	0.90	1.69	0.94
printtokens	0.76	1.17	1.33	3.49
schedule	0.71	1.04	1.19	2.62
printtokens2	0.73	1.13	1.29	3.41
schedule2	0.73	1.06	1.19	2.44
tcas	0.86	1.03	1.14	1.61
totinfo	0.91	1.16	1.35	1.97
flex	1.48	2.05	2.69	3.22
gzip	1.15	1.78	2.39	3.51
sed	1.05	1.80	2.70	6.71
space-1	1.78	2.83	3.98	6.28
space-2	1.82	2.88	4.03	6.41
space-3	1.80	2.86	4.06	6.45
space-4	1.77	2.86	3.98	6.18
replace	0.74	1.19	1.37	4.06
bash	1.56	2.93	4.88	22.96
ibm	3.13	5.72	9.29	24.62

Table 4 Speed-up results for TAEA

Subject	S_{JOMP1}	S_{JOMP2}	S_{JOMP4}	S_{GPU}
printtokens-1	0.73	1.19	1.68	2.60
printtokens-2	0.75	1.21	1.70	2.60
printtokens-3	0.73	1.18	1.66	2.61
printtokens-4	0.74	1.21	1.70	2.63
schedule-1	1.01	1.48	1.89	2.17
schedule-2	1.00	1.47	1.88	2.19
schedule-3	0.99	1.46	1.88	2.16
schedule-4	0.99	1.46	1.87	2.15
printtokens	0.47	0.82	0.98	4.58
schedule	0.49	0.84	1.03	3.94
printtokens2	0.47	0.83	1.00	4.63
schedule2	0.50	0.84	1.01	3.49
tcas	0.67	1.00	1.29	2.24
totinfo	0.68	1.09	1.54	2.99
flex	1.71	3.17	5.12	8.69
gzip	0.97	1.78	2.91	7.88
sed	0.85	1.60	2.66	10.85
space-1	1.79	3.29	5.33	12.01
space-2	1.83	3.39	5.53	12.51
space-3	1.79	3.33	5.49	12.21
space-4	1.77	3.31	5.43	11.93
replace	0.47	0.84	1.01	5.44
bash	0.88	1.69	2.89	17.71
ibm	2.06	3.87	6.54	20.97

Table 5 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

Table 6 Mann-Whitney U test for SPEA2

Subject	p_{JOMP1}	p_{JOMP2}	p_{JOMP4}	p_{GPU}
printtokens-1	9.99e−01	9.91e−01	8.71e−01	7.19e−01
printtokens-2	7.84e−01	9.81e−01	2.23e−01	3.06e−02
printtokens-3	9.39e−01	9.91e−01	9.85e−01	4.21e−01
printtokens-4	5.32e−01	9.85e−01	2.06e−01	3.71e−01
schedule-1	6.01e−01	1.00e+00	9.80e−01	9.01e−01
schedule-2	3.99e−02	9.99e−01	9.71e−01	3.27e−01
schedule-3	9.37e−01	1.00e+00	9.42e−01	9.59e−01
schedule-4	4.74e−01	1.00e+00	9.78e−01	9.94e−01
printtokens	1.00e+00	8.46e−18	1.50e−11	8.46e−18
schedule	1.00e+00	1.51e−11	1.50e−11	1.51e−11
printtokens2	1.00e+00	8.46e−18	1.50e−11	8.46e−18
schedule2	1.00e+00	1.51e−11	1.50e−11	1.51e−11
tcas	1.00e+00	1.51e−11	5.51e−07	1.51e−11
totinfo	1.00e+00	1.51e−11	1.50e−11	8.46e−18
flex	8.46e−18	8.46e−18	1.50e−11	8.46e−18
gzip	8.46e−18	8.46e−18	1.50e−11	8.46e−18
sed	8.46e−18	8.46e−18	1.50e−11	8.46e−18
space-1	8.46e−18	1.51e−11	1.50e−11	8.46e−18
space-2	8.46e−18	8.46e−18	1.50e−11	1.51e−11
space-3	8.46e−18	1.51e−11	1.50e−11	8.46e−18
space-4	8.46e−18	1.50e−11	1.50e−11	8.46e−18
replace	1.00e+00	1.50e−11	1.50e−11	8.46e−18
bash	8.46e−18	1.50e−11	1.50e−11	8.46e−18
ibm	8.46e−18	1.50e−11	1.50e−11	8.46e−18

Table 7 Mann-Whitney U test for TAEA

Subject	p_{JOMP1}	p_{JOMP2}	p_{JOMP4}	p_{GPU}
printtokens-1	1.00e+00	1.48e−11	1.50e−11	1.51e−11
printtokens-2	1.00e+00	1.50e−11	1.49e−11	1.51e−11
printtokens-3	1.00e+00	1.50e−11	1.50e−11	1.51e−11
printtokens-4	1.00e+00	1.49e−11	1.49e−11	1.51e−11
schedule-1	3.86e−02	1.48e−11	1.49e−11	1.51e−11
schedule-2	9.96e−01	1.49e−11	1.48e−11	1.50e−11
schedule-3	9.99e−01	1.50e−11	1.50e−11	1.51e−11
schedule-4	7.63e−01	1.50e−11	1.50e−11	1.50e−11
printtokens	1.00e+00	1.00e+00	1.00e+00	1.51e−11
schedule	1.00e+00	1.00e+00	3.66e−10	1.51e−11
printtokens2	1.00e+00	1.00e+00	5.85e−01	1.51e−11
schedule2	1.00e+00	1.00e+00	2.54e−06	1.51e−11
tcas	1.00e+00	4.50e−01	1.50e−11	1.51e−11
totinfo	1.00e+00	1.50e−11	1.50e−11	1.51e−11
flex	1.50e−11	1.50e−11	1.50e−11	1.50e−11
gzip	1.00e+00	1.50e−11	1.49e−11	1.51e−11
sed	1.00e+00	1.50e−11	1.50e−11	1.51e-11
space-1	1.50e−11	1.50e−11	1.50e−11	1.51e−11
space-2	1.50e−11	1.50e−11	1.50e−11	1.51e−11
space-3	1.50e−11	1.50e−11	1.50e−11	1.51e−11
space-4	1.50e−11	1.50e−11	1.50e−11	1.51e−11
replace	1.00e+00	1.00e+00	2.10e−02	1.51e−11
bash	1.00e+00	1.50e−11	1.50e−11	1.51e−11
ibm	1.50e−11	1.50e−11	1.50e−11	1.51e−11

confidence level of 95 %, respectively, out of of 42 cases with subjects with problem sizes smaller than that of `flex`, providing evidence that the parallel configurations required more time than the untreated configuration (`CPU`). With `JOMP4` and `GPU` configurations, the null hypothesis is universally rejected for all subjects with problem sizes larger than that of `flex`, 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 3.0×. This finding answers **RQ1**.

6.2 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 processors executes a single thread of Algorithm 1, of which 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 on speed-up would be 800. In practice, the individual processors run with a clock speed of 625 MHz, 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 constitute the size of problems.

Three size factors have been analysed for the statistical regression: size of test goal set, size of test suite and their product. The number of test requirements and the number of test cases are denoted by l and m respectively, following the matrix notation in Section 3: l is proportional to 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 8 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.

In all algorithms and configurations, the size factor l shows the strongest positive correlation with speed-ups. The correlation coefficients for z are weaker than those for l or, in the case of `JOMP1` for the Two Archive algorithm, negative. The correlation for m remains negative for all algorithms and configurations.

To gain further insights into the correlation between size and the speed-up observed, a regression analysis was performed. Factor z is considered in isolation, whereas l and m are considered together; each variable has been considered in

Table 8 Spearman's rank correlation coefficients between three size factors and speed-ups

Algorithm	Config	ρ_z	ρ_l	ρ_m
NSGA-II	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
SPEA2	JOMP1	0.2327	0.6646	−0.7827
	JOMP2	0.7897	0.7977	−0.3375
	JOMP4	0.6852	0.7201	−0.3286
	GPGPU	0.9022	0.7618	−0.1286
TAEA	JOMP1	−0.0084	0.5225	−0.9302
	JOMP2	0.1527	0.6580	−0.8867
	JOMP4	0.1671	0.6686	−0.8760
	GPGPU	0.8723	0.8729	−0.2536

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. Tables 9, 10 and 11 in the Appendix present the detailed results of regression analysis for the three algorithms respectively.

With a few exceptions of very small margins (NSGA-II with JOMP4 and SPEA2 with JOMP1, JOMP4, and GPU), the model with the highest r^2 correlation for all algorithms and configurations is $S_p = \alpha \log l + \beta \log m + \gamma$. Figure 2 shows the 3D plot of this model for the GPU and JOMP4 configuration of the Two Archive algorithm.

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 lower correlation.

Fig. 2 3D-plot of regression model $S_p = \alpha \log l + \beta \log m + \gamma$ for GPU (*solid line*) and JOMP4 (*dotted line*) configurations for two archive algorithm
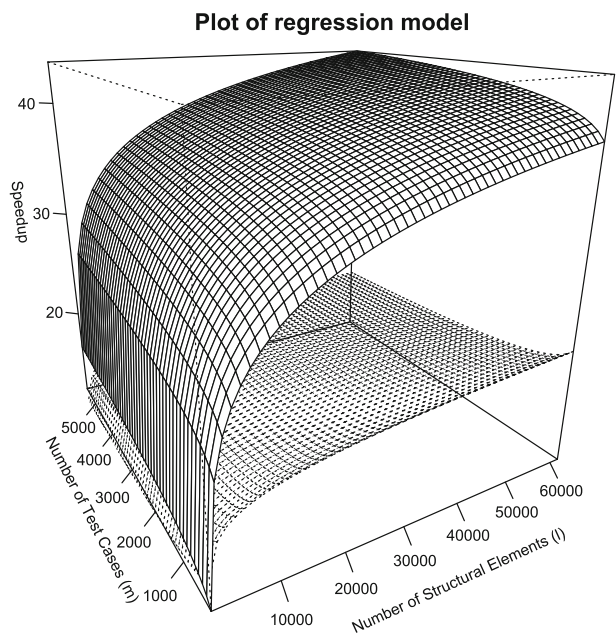
This supports the results of Spearman's rank correlation analysis in Table 8. The coefficients for the best-fit regression model for GPU, $S_p = \alpha \log l + \beta \log m + \gamma$, can explain why the speed-up results for space test suites are higher than those for test suites with z values such as tcas, gzip and replace. Apart from bash and ibm, space has the largest number of test requirements 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 requirements to cover and the log of the number of test cases in the test suite. The positive correlation provides 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, the speed-up would be even greater than those observed in this paper. The finding that the scalability factor increases with overall problem size is a very encouraging finding; as the problem gets harder, the solution gets better.

6.3 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 min of time to execute a subset of tests from a large test suite that would require more than 4 h to execute in its entirety.

Using static smoke test suite is problematic as running the same tests at every regression greatly reduces the likelihood of finding bugs. Therefore it is important to recalculate the most relevant smoke test suite given the changes to the code. It is for this reason that the cost of computation, especially the actual time it takes, becomes very important.

Figure 3 shows two possible smoke test scenarios based on the results of CPU and GPGPU configurations of NSGA-II. It is a plot of how much test requirement coverage can be achieved during the given time, including the time needed for multi-objective test suite minimisation. The solid line represents the scenario based on the GPGPU configuration of the algorithm, whereas the dotted line represents the scenario based on the CPU configuration. The beginning flat segment at the bottom 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 min to terminate, it cannot contribute to any smoke test scenario that must be completed within 60 min. On the other hand, the GPGPU configuration allows the tester to consider a subset of tests that can be executed within 30 min. If the grey region was wider than Fig. 3, 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 GPGPU are not only sources of efficiency improvement,

Fig. 3 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 min
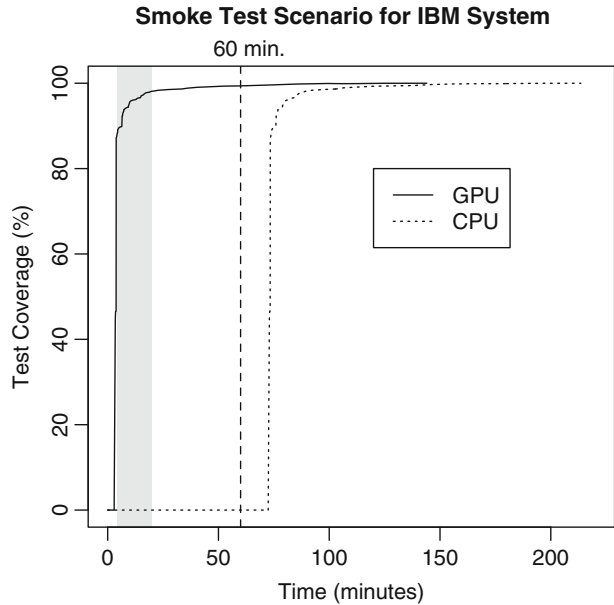
they can also make possible test activities that are simply impossible without this scalability.

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 (Yoo and Harman 2007) or the last time a specific test case has been executed (Kim and Porter 2002; Engström et al. 2010). 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 (Yoo et al. 2009).

7 Threats to Validity

Threats to internal validity concern the factors that could have affected the experiments in the paper. While GPGPU architecture has been researched for some time, the commercially available GPGPU frameworks such as CUDA and OpenCL are still in their early stages and, therefore, may contain faults in the implementation.

The GPGPU matrix-multiplication routine has been manually tested and validated with additional data apart from the test suites chosen for the empirical study. Regarding the precision of the GPGPU-based calculation, the aim of the paper is to investigate the potential speed-up that can be gained by using GPGPU, rather than

to consider the effectiveness of the actual test suite minimisation in the context of regression testing. Therefore, the precision issue does not constitute a major issue for the aim of this study.

Threats to external validity concern any factor that might prevent the generalisation of the result presented by the paper. Since the performance of GPGPU computing is inherently hardware specific, the results reported in the paper may not be reproducible in their exact form using other combinations of hardware components. However, with the advances in graphics card architecture, it is more likely that experiments with the same approach with newer graphics card will only improve the speed-up results reported in the paper.

It should be also noted that optimising test suite minimisation using evolutionary computation is an inherently ideal candidate for GPGPU computation as the reformulated problem, matrix-multiplication, is highly parallel in nature. Other problems in search-based software engineering may not render themselves as easily as the test suite minimisation problem. However, this issue is inherent in any attempts to parallelise a software engineering technique and not specific to GPGPU approach.

Threats to construct validity arise when measurements used in the experiments do not capture the concepts they are supposed to represent. The speed-up calculation was based on the measurements of execution time for both algorithms using system clock, which was chosen because it represents the *speed* of a technique to the end-user. Regarding the measurements of problem size used for the regression analysis, there may exist more sophisticated measurements of test suites and program source code that correlates better with the speed-up. However, both the number of test requirements and the number of test cases are probably the most readily available measurements about source code and test suites and provide a reasonable starting point for this type of analysis.

8 Related Work

Recent developments in graphics hardware provide an affordable means of parallelism: not only is the hardware more affordable than that of 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 (Boyer et al. 2009; Govindaraju et al. 2006), while Langdon and Banzhaf (2008) recently used GPGPU for performance improvements in optimization problems. However, these techniques have not been applied to Search Based Software Engineering problems, motivating the work of this paper, i,e., the use of GPGPU to achieve performance improvements in SBSE.

As a first instance of GPGPU SBSE, we study the SBSE application domain of regression testing. Regression test selection (also known as 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 (Harrold et al. 1993), which is NP-hard (Garey and Johnson 1979).

Various heuristics for the minimal representative set problem, or the minimal set cover problem (the dual of the former), have been suggested for test suite minimisation (Chen and Lau 1996; Offutt et al. 1995). However, empirical evaluations of these techniques have reported conflicting views on the impact on fault detection capability: some reported no impact (Wong et al. 1998, 1999) while others reported compromised fault detection capability (Rothermel et al. 2002a, b).

One potential reason why test suite minimisation has a 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 (2007); the paper also presents the first attempt to parallelise test suite minimisation with sophisticated criteria for scalability. Multi-objective forms of regression testing problems are increasingly popular in SBSE work, since most real-world regression testing scenarios need to satisfy multiple objectives (Harman 2011). Our SBSE approach to regression testing has also been used at Google (Yoo et al. 2011b).

Population-based evolutionary algorithms are ideal candidates for GPGPU parallelisation (Owens et al. 2007) 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) (Tsutsui and Fujimoto 2009). Wilson and Banzaf implemented a linear Genetic Programming (GP) algorithm on XBox360 game consoles (Wilson and Banzhaf 2009). Langdon and Banzaf implemented GP for GPU using an SIMD interpreter for fitness evaluation (Langdon and Banzhaf 2008). Wong implemented an MOEA on GPU and evaluated the implementation using a suite of benchmark problems (Wong 2009). 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. Of 763 papers on SBSE (Zhang 2011) only three present results for parallel execution of SBSE. Mitchell et al. used a distributed architecture for their clustering tool `Bunch` (Mitchell et al. 2001). Mahdavi et al. (2003) used a cluster of standard PCs to implement a parallel hill climbing algorithm. Asadi et al. used a distributed Server-Client architecture for Concept Location problem (Asadi et al. 2010). All three of these previous approaches use a distributed architecture that requires multiple machines. The present paper is the first work on SBSE that presents results for the highly affordable parallelism based on GPGPU.

There is existing work that re-implements meta-heuristic algorithms on GPGPU for non-SBSE applications. This previous work ports the meta-heuristic algorithms in their entirety (Wong 2009) to GPGPU, whereas the present paper concerns the practical improvement in scalability of SBSE, rather than the technical feasibility of GPGPU-based meta-heuristic implementation. The present paper thus re-implements only the most parallelisable module in Multi-Objective Evolutionary Algorithms (MOEAs) and performs an extensive empirical evaluation of the impact on scalability.

We believe that this approach may also prove to be applicable to many other SBSE problems. To achieve this, it will be necessary to develop new ways to port the fitness computation to the GPGPU device. For some applications, such as test data generation and re-generation problems (Ali et al. 2010; Yoo and Harman 2012b), this may not be possible, because the fitness function requires execution of the program under test; we cannot be sure that GPGPU devices will ever develop to the point that execution of arbitrary code will become possible.

However, for other SBSE applications, such as requirements optimisation (Saliu and Ruhe 2007; Zhang et al. 2011), prediction system feature selection (Kirsopp et al. 2002) and requirements sensitivity analysis (Harman et al. 2009), fitness computation remains an oft-repeated and, thereby, SIMD-friendly requirement. Also, in these application areas, the underlying Software Engineering problem may be characterised using a table (albeit a very large one). In such cases, where the SBSE problem can be formulated in terms of the optimisation of choices, based on a spreadsheet of known values, this may prove to port well onto the SIMD architecture offered by GPGPU devices.

Furthermore, for requirements optimisation problems there is known to be a close similarity between the problem representation for search based requirements and search based regression testing (Harman et al. 2012). As a result, the techniques used here may also prove to be readily applicable to these problems with little need for significant modification of our approach.

9 Conclusion

This paper presented the first results on GPGPU SBSE; the 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 provide an ideal platform for improving SBSE scalability through SIMD parallelism.

The paper presents an evaluation of the GPGPU-based test suite minimisation for real-world examples that include an industry-scale test suite. This approach to GPGPU SBSE was evaluated on three popular multi-objective evolutionary algorithms. The results show that the GPGPU-based optimisation can achieve a speed-up of up to $25.09\times$ 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 $9.29\times$. 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. Future work will include an empirical study of a wider range of test suites, as well as seeking insights into why MOEAs benefit differently from parallelisation.

Appendix

Tables 9, 10 and 11 present the results of regression analysis for the three algorithm respectively. Tables 12, 13 and 14 contain the mean and standard deviation of $Time_{total}$, $Time_{init}$, $Time_{fitness}$ and $Time_{remaining}$ for NSGA-II, SPEA2 and Two Archive algorithm respectively.

Table 9 Regression analysis for NSGA-II

Config	Model	α	β	γ	R^2
JOMP 1	$S_p \sim z$	1.56e−07	–	1.00e+00	0.4894
	$S_p \sim \log z$	2.01e−01	–	−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
JOMP 2	$S_p \sim z$	3.24e−07	–	1.58e+00	0.5009
	$S_p \sim \log z$	4.78e−01	–	−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
JOMP 4	$S_p \sim z$	5.80e−07	–	2.15e+00	0.5045
	$S_p \sim \log z$	8.72e−01	–	−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	–	4.13e+00	0.7261
	$S_p \sim \log z$	3.45e+00	–	−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

Table 10 Regression analysis for SPEA2

Config	Model	α	β	γ	R^2
JOMP 1	$S_p \sim z$	1.60e−07	–	1.03e+00	0.5085
	$S_p \sim \log z$	1.89e−01	–	−1.16e+00	0.2988
	$S_p \sim l + m$	3.37e−05	−1.20e−04	1.21e+00	0.7443
	$S_p \sim \log l + m$	2.58e−01	−6.08e−05	−3.57e−01	0.7987
	$S_p \sim l + \log m$	3.23e−05	−1.79e−01	2.19e+00	0.7883
	$S_p \sim \log l + \log m$	2.50e−01	−7.97e−02	1.17e−01	0.7982

Table 10 (continued)

Config	Model	α	β	γ	R^2
JOMP2	$S_p \sim z$	3.67e−07	–	1.31e+00	0.6289
	$S_p \sim \log z$	5.31e−01	–	−4.94e+00	0.5567
	$S_p \sim l + m$	7.41e−05	−1.02e−04	1.53e+00	0.6867
	$S_p \sim \log l + m$	6.14e−01	4.59e−05	−2.22e+00	0.8656
	$S_p \sim l + \log m$	7.24e−05	−1.78e−01	2.52e+00	0.7031
	$S_p \sim \log l + \log m$	6.30e−01	9.28e−02	−2.85e+00	0.8700
JOMP4	$S_p \sim z$	6.26e−07	–	1.78e+00	0.5504
	$S_p \sim \log z$	7.86e−01	–	−7.37e+00	0.3657
	$S_p \sim l + m$	1.23e−04	−2.40e−04	2.25e+00	0.5965
	$S_p \sim \log l + m$	9.38e−01	−2.73e−05	−3.44e+00	0.6443
	$S_p \sim l + \log m$	1.20e−04	−3.56e−01	4.19e+00	0.6081
	$S_p \sim \log l + \log m$	9.56e−01	3.15e−02	−3.78e+00	0.6442
GPU	$S_p \sim z$	2.32e−06	–	2.25e+00	0.8777
	$S_p \sim \log z$	3.12e+00	–	−3.42e+01	0.6666
	$S_p \sim l + m$	3.82e−04	1.98e−04	3.06e+00	0.5713
	$S_p \sim \log l + m$	3.01e+00	8.99e−04	−1.52e+01	0.6657
	$S_p \sim l + \log m$	3.90e−04	5.17e−01	4.89e−02	0.5791
	$S_p \sim \log l + \log m$	3.38e+00	1.96e+00	−2.88e+01	0.7417

Table 11 Regression analysis for two archive

Config	Model	α	β	γ	R^2
JOMP1	$S_p \sim z$	7.34e−08	–	9.35e−01	0.1280
	$S_p \sim \log z$	9.65e−02	–	−1.92e−01	0.0931
	$S_p \sim l + m$	1.78e−05	−1.74e−04	1.14e+00	0.5412
	$S_p \sim \log l + m$	1.94e−01	−1.20e−04	−7.59e−02	0.7637
	$S_p \sim l + \log m$	1.54e−05	−2.79e−01	2.68e+00	0.7108
	$S_p \sim \log l + \log m$	1.64e−01	−2.01e−01	1.22e+00	0.8350
JOMP2	$S_p \sim z$	1.60e−07	–	1.59e+00	0.1587
	$S_p \sim \log z$	2.57e−01	–	−1.45e+00	0.1731
	$S_p \sim l + m$	3.72e−05	−2.98e−04	1.95e+00	0.4942
	$S_p \sim \log l + m$	4.31e−01	−1.73e−04	−7.67e−01	0.8095
	$S_p \sim l + \log m$	3.27e−05	−4.94e−01	4.69e+00	0.6461
	$S_p \sim \log l + \log m$	3.84e−01	−3.04e−01	1.22e+00	0.8571
JOMP4	$S_p \sim z$	3.12e−07	–	2.33e+00	0.1865
	$S_p \sim \log z$	5.21e−01	–	−3.84e+00	0.2196
	$S_p \sim l + m$	6.95e−05	−5.20e−04	2.97e+00	0.4990
	$S_p \sim \log l + m$	8.17e−01	−2.82e−04	−2.18e+00	0.8556
	$S_p \sim l + \log m$	6.17e−05	−8.50e−01	7.69e+00	0.6322
	$S_p \sim \log l + \log m$	7.46e−01	−4.77e−01	9.01e−01	0.8880
GPU	$S_p \sim z$	1.64e−06	–	4.96e+00	0.5728
	$S_p \sim \log z$	2.79e+00	–	−2.82e+01	0.7056
	$S_p \sim l + m$	2.83e−04	−3.54e−04	6.02e+00	0.4516
	$S_p \sim \log l + m$	3.05e+00	5.02e−04	−1.31e+01	0.9417
	$S_p \sim l + \log m$	2.76e−04	−6.36e−01	9.59e+00	0.4620
	$S_p \sim \log l + \log m$	3.21e+00	9.47e−01	−1.94e+01	0.9603

Table 12 Execution time of NSGA-II algotirhm

Subject	Config	$\bar{T}_{total}$	σT_{total}	$\bar{T}_{init}$	σT_{init}	$\bar{T}_{fitness}$	$\sigma T_{fitness}$	$\bar{T}_{remaining}$	$\sigma T_{remaining}$
printtokens-1	CPU	12,265.23	133.47	0.00	0.00	8,565.77	63.68	3,699.47	104.78
printtokens-1	JOMP1	14,869.77	379.38	5.83	0.45	11,084.23	310.38	3,779.70	119.98
printtokens-1	JOMP2	10,112.50	146.97	5.70	0.46	5,905.90	87.01	4,200.90	107.54
printtokens-1	JOMP4	7,950.77	165.02	5.67	0.47	3,633.93	63.73	4,311.17	127.42
printtokens-1	GPGPU	5,739.60	145.89	469.33	3.64	1,934.33	124.20	3,335.93	100.65
printtokens-2	CPU	12,518.40	146.89	0.00	0.00	8,756.17	68.64	3,762.23	109.13
printtokens-2	JOMP1	15,029.73	383.27	5.77	0.62	11,220.00	297.28	3,803.97	120.04
printtokens-2	JOMP2	10,162.13	140.35	5.73	0.44	5,954.07	88.72	4,202.33	102.93
printtokens-2	JOMP4	8,036.67	131.23	5.80	0.40	3,692.87	68.79	4,338.00	103.05
printtokens-2	GPGPU	5,685.50	146.80	468.90	2.20	1,867.03	109.29	3,349.57	95.57
printtokens-3	CPU	12,335.80	131.26	0.00	0.00	8,626.63	60.42	3,709.17	102.36
printtokens-3	JOMP1	14,965.80	416.84	5.80	0.48	11,179.60	297.77	3,780.40	156.59
printtokens-3	JOMP2	10,185.57	128.75	5.53	0.50	5,930.50	73.85	4,249.53	102.33
printtokens-3	JOMP4	8,086.13	167.18	5.77	0.42	3,688.27	84.40	4,392.10	112.38
printtokens-3	GPGPU	5,784.47	157.36	468.33	4.78	1,952.60	127.56	3,363.53	82.57
printtokens-4	CPU	12,360.47	115.52	0.00	0.00	8,670.77	62.58	3,689.70	81.87
printtokens-4	JOMP1	14,690.43	337.86	5.73	0.44	11,004.50	264.70	3,680.20	141.57
printtokens-4	JOMP2	10,140.37	115.35	5.70	0.46	5,944.17	85.72	4,190.50	104.89
printtokens-4	JOMP4	8,010.47	199.55	5.63	0.48	3,673.30	129.08	4,331.53	129.24
printtokens-4	GPGPU	5,642.50	157.42	467.80	2.91	1,883.70	139.35	3,291.00	90.03
schedule-1	CPU	7,638.67	117.48	0.00	0.00	4,439.20	70.69	3,199.47	65.11
schedule-1	JOMP1	7,871.10	107.36	3.40	0.49	4,679.13	86.89	3,188.57	82.29
schedule-1	JOMP2	6,257.03	90.83	3.47	0.50	2,630.67	54.54	3,622.90	105.58
schedule-1	JOMP4	5,450.87	97.13	3.57	0.50	1,740.70	43.13	3,706.60	84.68
schedule-1	GPGPU	4,893.73	219.03	475.37	2.50	1,631.47	213.80	2,786.90	82.99

Table 12 (continued)

Subject	Config	$\bar{T}_{\text{total}}$	$\sigma_{T_{\text{total}}}$	$\bar{T}_{\text{init}}$	$\sigma_{T_{\text{init}}}$	$\bar{T}_{\text{fitness}}$	$\sigma_{T_{\text{fitness}}}$	$\bar{T}_{\text{remaining}}$	$\sigma_{T_{\text{remaining}}}$
schedule-2	CPU	7,745.30	104.94	0.00	0.00	4,499.87	39.01	3,245.43	80.30
schedule-2	JOMP1	8,032.80	113.87	3.53	0.50	4,793.07	75.86	3,236.20	90.57
schedule-2	JOMP2	6,341.13	111.05	3.60	0.49	2,676.43	49.23	3,661.10	97.28
schedule-2	JOMP4	5,504.40	141.96	3.47	0.50	1,760.17	57.80	3,740.77	104.00
schedule-2	GPGPU	5,304.50	112.85	474.90	2.17	2,028.83	21.46	2,800.77	93.53
schedule-3	CPU	7,646.40	124.66	0.00	0.00	4,461.60	53.81	3,184.80	89.71
schedule-3	JOMP1	7,941.90	129.85	3.47	0.50	4,715.47	99.74	3,222.97	92.20
schedule-3	JOMP2	6,251.20	95.49	3.47	0.50	2,632.60	39.48	3,615.13	96.39
schedule-3	JOMP4	5,509.93	125.49	3.60	0.49	1,750.40	50.08	3,755.93	92.32
schedule-3	GPGPU	5,285.13	120.56	474.57	1.56	2,026.90	19.19	2,783.67	104.82
schedule-4	CPU	7,611.70	92.16	0.00	0.00	4,430.17	41.45	3,181.53	69.16
schedule-4	JOMP1	8,033.37	122.39	3.47	0.50	4,792.00	92.38	3,237.90	96.45
schedule-4	JOMP2	6,359.90	85.07	3.63	0.48	2,693.93	45.06	3,662.33	84.43
schedule-4	JOMP4	5,553.03	100.72	3.53	0.50	1,771.70	38.32	3,777.80	88.11
schedule-4	GPGPU	5,307.77	112.28	474.83	1.85	2,037.33	20.37	2,795.60	96.75
printtokens	CPU	201,468.50	1,017.39	0.00	0.00	168,824.77	933.12	32,643.73	217.17
printtokens	JOMP1	264,294.97	730.51	12.20	0.40	231,541.57	668.03	32,741.20	268.03
printtokens	JOMP2	162,367.67	368.62	12.47	0.50	124,352.20	351.64	38,003.00	298.90
printtokens	JOMP4	140,384.07	319.11	12.23	0.42	102,300.67	184.97	38,071.17	242.94
printtokens	GPGPU	44,592.67	234.10	470.10	1.35	12,097.70	26.65	32,024.87	234.80
schedule	CPU	95,693.77	607.90	0.00	0.00	74,140.63	504.25	21,553.13	175.79
schedule	JOMP1	139,348.20	609.53	16.73	0.51	117,751.40	547.86	21,580.07	310.49
schedule	JOMP2	88,385.17	383.10	16.53	0.50	63,433.77	271.22	24,934.87	304.31
schedule	JOMP4	75,779.67	584.89	16.63	0.55	50,686.53	377.03	25,076.50	480.73
schedule	GPGPU	28,351.07	324.36	464.73	1.59	6,899.33	20.27	20,987.00	328.10

Table 12 (continued)

Subject	Config	$\bar{T}_{total}$	σT_{total}	$\bar{T}_{init}$	σT_{init}	$\bar{T}_{fitness}$	$\sigma T_{fitness}$	$\bar{T}_{remaining}$	$\sigma T_{remaining}$
printtokens2	CPU	200,409.53	1,007.63	0.00	0.00	167,983.10	860.30	32,426.43	256.84
printtokens2	JOMP1	278,160.67	788.57	12.57	0.50	245,605.30	794.70	32,542.80	217.64
printtokens2	JOMP2	169,781.93	604.65	12.33	0.54	132,011.97	481.70	37,757.63	340.18
printtokens2	JOMP4	146,077.10	460.93	12.43	0.50	108,003.00	325.04	38,061.67	335.96
printtokens2	GPGPU	45,705.40	221.84	470.67	1.30	13,294.90	27.15	31,939.83	219.62
schedule2	CPU	88,307.47	907.51	0.00	0.00	66,683.77	728.58	21,623.70	409.08
schedule2	JOMP1	124,601.87	585.05	16.33	0.60	102,931.23	557.67	21,654.30	409.55
schedule2	JOMP2	80,791.20	587.41	16.70	0.97	55,709.73	231.85	25,064.77	491.77
schedule2	JOMP4	69,575.73	536.93	16.50	1.06	44,214.20	299.20	25,345.03	424.66
schedule2	GPGPU	28,571.07	359.22	462.53	1.80	6,794.00	21.28	21,314.53	363.45
tcas	CPU	33,098.07	403.64	0.00	0.00	19,479.47	335.46	13,618.60	126.26
tcas	JOMP1	39,282.17	423.11	14.07	1.31	25,542.43	408.53	13,725.67	178.59
tcas	JOMP2	30,021.70	308.38	14.20	1.30	14,239.20	215.67	15,768.30	189.47
tcas	JOMP4	25,391.43	229.51	14.10	1.47	9,460.17	189.81	15,917.17	242.64
tcas	GPGPU	17,099.93	166.20	466.40	2.32	3,476.97	22.49	13,156.57	163.79
totinfo	CPU	33,547.10	414.49	0.00	0.00	23,190.27	212.64	10,356.83	236.96
totinfo	JOMP1	37,089.93	305.02	13.23	1.33	26,853.87	285.81	10,222.83	146.45
totinfo	JOMP2	26,280.27	277.43	13.43	1.17	14,567.70	155.58	11,699.13	196.05
totinfo	JOMP4	20,867.60	208.74	13.70	1.59	8,988.33	76.84	11,865.57	189.78
totinfo	GPGPU	13,409.37	131.48	465.00	3.04	3,065.73	28.02	9,878.63	132.19
flex	CPU	68,898.23	562.97	0.00	0.00	6,6074.40	554.00	2,823.83	60.97
flex	JOMP1	43,761.27	1,667.94	14.07	0.73	40,925.87	1,678.87	2,821.33	74.69
flex	JOMP2	24,966.77	513.57	13.63	0.48	21,769.90	509.27	3,183.23	60.48
flex	JOMP4	16,441.23	232.20	13.90	0.54	13,223.03	231.11	3,204.30	71.58
flex	GPGPU	10,103.20	65.62	465.27	2.31	7,225.33	10.47	2,412.60	66.89

Table 12 (continued)

Subject	Config	$\bar{T}_{\text{total}}$	$\sigma_{T_{\text{total}}}$	$\bar{T}_{\text{init}}$	$\sigma_{T_{\text{init}}}$	$\bar{T}_{\text{fitness}}$	$\sigma_{T_{\text{fitness}}}$	$\bar{T}_{\text{remaining}}$	$\sigma_{T_{\text{remaining}}}$
gzip	CPU	73,950.87	959.62	0.00	0.00	70,627.90	946.41	3,322.97	58.53
gzip	JOMP1	62,003.70	1,154.83	12.77	0.76	58,680.57	1,157.14	3,310.37	65.29
gzip	JOMP2	34,440.27	591.68	12.40	0.55	30,655.77	582.96	3,772.10	87.57
gzip	JOMP4	22,367.40	539.74	12.57	0.80	18,535.70	545.24	3,819.13	82.22
gzip	GPGPU	9,240.03	69.80	463.90	1.70	5,831.23	9.23	2,944.90	69.46
sed	CPU	124,817.33	1,976.92	0.00	0.00	120,265.57	1,930.36	4,551.77	72.83
sed	JOMP1	122,040.30	1,435.61	11.73	0.57	117,454.93	1,441.36	4,573.63	68.62
sed	JOMP2	66,612.53	649.39	11.53	0.56	61,453.23	645.79	5,147.77	82.37
sed	JOMP4	41,056.63	385.78	11.47	0.56	35,821.10	398.33	5,224.07	81.48
sed	GPGPU	12,147.77	75.00	467.27	2.14	7,498.07	11.22	4,182.43	77.26
space-1	CPU	128,911.03	3,000.98	0.00	0.00	125,323.07	2,979.24	3,587.97	62.34
space-1	JOMP1	72,884.27	1,545.06	13.27	0.81	69,262.87	1,538.28	3,608.13	74.12
space-1	JOMP2	39,989.00	878.97	13.13	0.43	35,861.10	859.12	4,114.77	89.34
space-1	JOMP4	25,293.07	395.23	13.20	0.40	21,164.57	392.79	4,115.30	86.67
space-1	GPGPU	12,270.57	61.69	466.80	1.80	8,622.20	10.40	3,181.57	64.01
space-2	CPU	126,462.67	2,652.95	0.00	0.00	122,919.20	2,592.98	3,543.47	87.69
space-2	JOMP1	68,066.23	1,147.03	13.07	0.44	64,527.43	1,164.34	3,525.73	78.28
space-2	JOMP2	37,911.63	594.27	13.17	0.37	33,840.57	583.31	4,057.90	71.06
space-2	JOMP4	24,380.70	555.05	13.00	0.26	20,286.67	553.22	4,081.03	69.44
space-2	GPGPU	11,625.40	68.16	465.10	1.62	8,021.33	9.84	3,138.97	68.86
space-3	CPU	130,576.67	2,677.40	0.00	0.00	126,974.30	2,640.58	3,602.37	73.72
space-3	JOMP1	72,470.93	1,543.13	13.03	0.31	68,864.00	1,531.11	3,593.90	75.07
space-3	JOMP2	39,988.90	784.99	13.10	0.54	35,870.73	777.28	4,105.07	78.43
space-3	JOMP4	25,302.80	447.97	13.20	0.40	21,153.63	433.21	4,135.97	74.92
space-3	GPGPU	12,279.10	84.11	466.67	1.94	8,622.53	8.05	3,189.90	86.12

Table 12 (continued)

Subject	Config	$\bar{T}_{total}$	σT_{total}	$\bar{T}_{init}$	σT_{init}	$\bar{T}_{fitness}$	$\sigma T_{fitness}$	$\bar{T}_{remaining}$	$\sigma T_{remaining}$
space-4	CPU	128,981.73	3,442.49	0.00	0.00	125,395.00	3,394.39	3,586.73	78.57
space-4	JOMP1	73,208.10	2,310.12	13.10	0.30	69,642.43	2,325.78	3,552.57	61.09
space-4	JOMP2	39,689.37	800.83	13.13	0.34	35,634.33	818.95	4,041.90	91.29
space-4	JOMP4	25,216.80	351.18	13.07	0.36	21,115.67	332.19	4,088.07	82.16
space-4	GPGPU	12,233.17	81.10	466.30	1.73	8,622.07	10.74	3,144.80	80.37
replace	CPU	325,246.37	1,698.49	0.00	0.00	281,927.93	1,405.19	43,318.43	848.20
replace	JOMP1	445,375.07	1,524.35	13.30	0.46	402,127.67	1,236.82	43,234.10	835.37
replace	JOMP2	265,138.93	1,078.85	13.20	0.48	214,949.63	672.27	50,176.10	848.70
replace	JOMP4	225,739.00	892.89	13.20	0.48	175,134.70	253.04	50,591.10	888.86
replace	GPGPU	61,807.93	519.29	472.07	3.92	18,291.03	31.20	43,044.83	523.68
bash	CPU	2,071,836.07	29,845.30	0.00	0.00	2,051,591.57	29,674.30	20,244.50	206.06
bash	JOMP1	1,346,585.83	16,962.75	53.30	1.39	1,326,396.90	16,966.54	20,135.63	159.67
bash	JOMP2	715,605.03	11,951.22	53.37	1.25	693,778.67	11,979.47	21,773.00	150.21
bash	JOMP4	425,783.60	5,673.48	54.07	1.73	403,970.63	5,677.03	21,758.90	209.60
bash	GPGPU	82,574.53	194.61	517.07	2.35	62,371.57	13.89	19,685.90	189.82
ibm	CPU	4,347,517.13	462,072.40	0.00	0.00	4,178,547.93	831,883.88	168,969.20	709,728.80
ibm	JOMP1	1,445,294.57	38,625.23	136.07	3.59	1,406,525.60	39,448.65	38,632.90	5,916.45
ibm	JOMP2	783,762.87	20,494.64	136.13	3.48	745,728.40	20,522.99	37,898.33	3,591.78
ibm	JOMP4	481,433.27	11,823.74	135.63	3.77	444,301.60	11,749.65	36,996.03	784.43
ibm	GPGPU	174,990.80	5,095.10	613.67	61.27	136,661.40	849.63	37,715.73	4,931.15

Table 13 Execution time of SPEA2 algotirhm

Subject	Config	$\bar{T}_{\text{total}}$	$\sigma_{T_{\text{total}}}$	$\bar{T}_{\text{init}}$	$\sigma_{T_{\text{init}}}$	$\bar{T}_{\text{fitness}}$	$\sigma_{T_{\text{fitness}}}$	$\bar{T}_{\text{remaining}}$	$\sigma_{T_{\text{remaining}}}$
printtokens-1	CPU	54,737.30	7,183.20	0.00	0.00	8,562.97	75.10	46,174.33	7,167.90
printtokens-1	JOMP1	60,409.20	7,729.38	5.73	0.44	10,825.50	121.39	49,577.97	7,709.26
printtokens-1	JOMP2	59,056.57	7,209.10	5.73	0.44	5,840.13	69.33	53,210.70	7,212.29
printtokens-1	JOMP4	52,838.37	15,410.19	5.77	0.42	3,571.33	39.62	49,261.27	15,434.46
printtokens-1	GPGPU	55,810.83	7,876.04	471.83	14.16	3,535.63	35.46	51,803.37	7,864.52
printtokens-2	CPU	61,021.80	9,959.34	0.00	0.00	8,769.17	68.85	52,252.63	9,922.10
printtokens-2	JOMP1	61,762.40	7,379.62	5.80	0.40	11,007.90	168.96	50,748.70	7,391.38
printtokens-2	JOMP2	67,094.07	12,558.61	5.77	0.42	5,971.47	76.16	61,116.83	12,560.31
printtokens-2	JOMP4	54,581.23	15,862.40	5.90	0.30	3,634.47	68.27	50,940.87	15,904.60
printtokens-2	GPGPU	56,347.47	8,010.39	468.90	2.36	3,521.87	30.39	52,356.70	8,007.62
printtokens-3	CPU	55,246.60	8,305.32	0.00	0.00	8,658.43	66.02	46,588.17	8,299.09
printtokens-3	JOMP1	57,619.10	6,555.12	5.83	0.37	10,936.97	133.77	46,676.30	6,526.33
printtokens-3	JOMP2	60,952.13	11,448.43	5.63	0.48	5,931.23	59.59	55,015.27	11,439.49
printtokens-3	JOMP4	57,878.43	19,310.31	5.67	0.47	3,601.83	44.26	54,270.93	19,337.38
printtokens-3	GPGPU	54,563.63	7,163.49	467.97	1.83	3,530.53	22.73	50,565.13	7,159.91
printtokens-4	CPU	59,433.97	7,999.19	0.00	0.00	8,692.00	70.85	50,741.97	7,986.66
printtokens-4	JOMP1	59,541.43	6,497.08	5.67	0.47	10,950.07	156.36	48,585.70	6,514.42
printtokens-4	JOMP2	64,436.50	8,461.38	5.77	0.42	5,915.53	64.97	58,515.20	8,458.26
printtokens-4	JOMP4	56,524.23	20,522.42	5.67	0.47	3,590.87	51.32	52,927.70	20,535.99
printtokens-4	GPGPU	58,235.23	5,732.37	467.17	2.38	3,101.50	570.40	54,666.57	5,759.73
schedule-1	CPU	103,525.27	10,751.91	0.00	0.00	4,445.07	31.51	99,080.20	10,752.81
schedule-1	JOMP1	104,730.07	13,222.87	3.70	0.46	4,700.47	109.48	100,025.90	13,203.26
schedule-1	JOMP2	115,522.53	9,903.85	3.63	0.48	2,618.30	49.96	112,900.60	9,911.08
schedule-1	JOMP4	100,832.93	35,751.09	3.77	0.42	1,669.10	31.57	99,160.07	35,770.11
schedule-1	GPGPU	108,538.97	13,846.59	475.30	1.66	2,570.70	165.46	105,492.97	13,856.20

Table 13 (continued)

Subject	Config	$\bar{T}_{\text{total}}$	$\sigma_{T_{\text{total}}}$	$\bar{T}_{\text{init}}$	$\sigma_{T_{\text{init}}}$	$\bar{T}_{\text{fitness}}$	$\sigma_{T_{\text{fitness}}}$	$\bar{T}_{\text{remaining}}$	$\sigma_{T_{\text{remaining}}}$
schedule-2	CPU	111,070.57	8,120.00	0.00	0.00	4,522.30	32.82	106,548.27	8,118.41
schedule-2	JOMP1	107,589.33	8,283.42	3.70	0.46	4,799.63	61.40	102,786.00	8,297.07
schedule-2	JOMP2	117,569.23	9,742.41	3.57	0.50	2,684.60	48.63	114,881.07	9,751.08
schedule-2	JOMP4	106,101.20	37,473.79	3.40	0.49	1,694.33	39.42	104,403.47	37,495.24
schedule-2	GPGPU	110,804.83	9,195.29	476.23	2.68	2,616.37	10.64	107,712.23	9,196.71
schedule-3	CPU	67,744.33	8,141.45	0.00	0.00	4,479.80	34.50	63,264.53	8,158.15
schedule-3	JOMP1	71,534.80	8,905.75	3.67	0.47	4,676.60	68.23	66,854.53	8,912.32
schedule-3	JOMP2	77,546.20	9,676.54	3.47	0.50	2,624.80	59.67	74,917.93	9,667.77
schedule-3	JOMP4	67,597.97	24,248.38	3.60	0.49	1,665.37	35.18	65,929.00	24,263.71
schedule-3	GPGPU	71,874.93	8,807.73	474.87	1.48	2,621.33	8.62	68,778.73	8,808.51
schedule-4	CPU	83,107.30	9,619.47	0.00	0.00	4,449.13	40.74	78,658.17	9,599.56
schedule-4	JOMP1	83,825.20	11,900.34	3.67	0.47	4,780.03	83.15	79,041.50	11,903.98
schedule-4	JOMP2	93,997.57	13,035.56	3.57	0.50	2,658.10	56.89	91,335.90	13,040.25
schedule-4	JOMP4	83,453.57	30,179.48	3.73	0.44	1,682.83	32.68	81,767.00	30,193.44
schedule-4	GPGPU	88,935.70	9,241.76	474.93	2.06	2,622.30	10.36	85,838.47	9,240.44
printtokens	CPU	218,854.33	1,726.93	0.00	0.00	168,196.60	1,514.85	50,657.73	327.32
printtokens	JOMP1	28.7307.47	1,127.93	12.37	0.55	236,604.07	929.05	50,691.03	355.26
printtokens	JOMP2	186,358.63	576.28	12.47	0.85	127,034.93	346.14	59,311.23	386.97
printtokens	JOMP4	164,231.80	2,976.55	12.20	0.40	103,544.77	472.16	60,674.83	2,558.07
printtokens	GPGPU	62,718.07	617.76	470.47	1.50	12,305.87	27.83	49,941.73	617.07
schedule	CPU	108,266.97	707.22	0.00	0.00	73,895.10	578.85	34,371.87	213.91
schedule	JOMP1	152,762.60	743.22	16.60	0.49	118,391.37	599.61	34,354.63	549.03
schedule	JOMP2	104,064.37	734.29	16.43	0.50	63,909.73	307.38	40,138.20	679.72
schedule	JOMP4	91,078.00	2,702.94	16.57	0.50	50,229.77	412.43	40,831.67	2,467.27
schedule	GPGPU	41,389.13	616.10	464.77	1.12	7,065.23	24.31	33,859.13	621.14

Table 13 (continued)

Subject	Config	$\bar{T}_{\text{total}}$	σT_{total}	$\bar{T}_{\text{init}}$	σT_{init}	$\bar{T}_{\text{fitness}}$	$\sigma T_{\text{fitness}}$	$\bar{T}_{\text{remaining}}$	$\sigma T_{\text{remaining}}$
printtokens2	CPU	218,065.40	885.64	0.00	0.00	167,678.67	670.63	50,386.73	385.17
printtokens2	JOMP1	298,648.77	940.76	12.43	0.62	248,103.87	788.46	50,532.47	367.08
printtokens2	JOMP2	192,170.00	478.57	12.47	0.50	132,948.23	405.78	59,209.30	293.85
printtokens2	JOMP4	168,897.87	2,981.31	12.17	0.37	108,599.40	488.40	60,286.30	2,579.16
printtokens2	GPGPU	63,975.20	273.85	485.47	79.59	13,622.70	27.21	49,867.03	313.69
schedule2	CPU	101,383.53	1,208.80	0.00	0.00	67,082.43	1,167.92	34,301.10	624.25
schedule2	JOMP1	138,615.47	2,661.56	16.73	1.09	103,845.77	1,939.76	34,752.97	918.69
schedule2	JOMP2	95,390.57	928.84	16.67	1.11	55,567.10	200.16	39,806.80	881.44
schedule2	JOMP4	85,278.97	2,484.97	16.37	0.60	44,539.50	439.27	40,723.10	2,316.45
schedule2	GPGPU	41,635.23	648.52	462.50	1.41	6,981.63	23.46	34,191.10	641.56
tcas	CPU	41,985.83	338.61	0.00	0.00	19,457.23	195.99	22,528.60	207.74
tcas	JOMP1	48,657.30	796.67	14.10	1.37	25,970.57	431.74	22,672.63	412.30
tcas	JOMP2	40,586.10	295.08	13.70	1.16	14,255.30	214.12	26,317.10	227.65
tcas	JOMP4	37,024.63	2,397.64	14.30	1.00	9,625.57	222.57	27,384.77	2,238.37
tcas	GPGPU	26,093.17	226.88	465.57	2.73	3,544.30	63.69	22,083.30	188.19
totinfo	CPU	40,370.83	424.62	0.00	0.00	23,053.97	205.96	17,316.87	250.47
totinfo	JOMP1	44,349.27	366.13	13.47	1.33	27,013.60	233.96	17,322.20	259.02
totinfo	JOMP2	34,824.07	391.81	13.10	1.11	14,644.00	189.26	20,166.97	285.14
totinfo	JOMP4	29,982.03	1,869.13	13.63	1.76	8,985.83	155.63	20,982.57	1,745.97
totinfo	GPGPU	20,475.40	242.92	464.03	1.96	3,071.80	56.35	16,939.57	222.17
flex	CPU	85,954.07	1,862.67	0.00	0.00	67,851.50	638.68	18,102.57	1,641.89
flex	JOMP1	57,963.13	1,777.55	13.83	0.64	40,302.90	1,188.90	17,646.40	1,186.56
flex	JOMP2	41,961.70	1,604.81	13.90	0.83	21,351.73	522.14	20,596.07	1,538.22
flex	JOMP4	32,708.03	4,129.65	14.30	0.59	12,716.27	304.51	19,977.47	4,062.74
flex	GPGPU	26,750.37	1,510.63	465.13	1.96	7,324.43	28.65	18,960.80	1,490.76

Table 13 (continued)

Subject	Config	$\bar{T}_{total}$	$\sigma_{T_{total}}$	$\bar{T}_{init}$	$\sigma_{T_{init}}$	$\bar{T}_{fitness}$	$\sigma_{T_{fitness}}$	$\bar{T}_{remaining}$	$\sigma_{T_{remaining}}$
gzip	CPU	90,650.70	1,740.26	0.00	0.00	73,402.77	908.02	17,247.93	1,519.11
gzip	JOMP1	78,942.40	2,740.94	13.37	2.85	60,347.33	1,754.09	18,581.70	1,995.90
gzip	JOMP2	51,022.47	1,655.06	12.67	0.54	30,961.97	595.97	20,047.83	1,539.96
gzip	JOMP4	38,451.73	4,173.62	12.40	0.49	18,194.50	454.46	20,244.83	4,111.11
gzip	GPGPU	25,936.73	1,699.42	464.23	2.03	5,960.30	22.46	19,512.20	1,683.50
sed	CPU	136,519.00	1,517.32	0.00	0.00	123,618.43	1,653.37	12,900.57	1,057.79
sed	JOMP1	129,451.17	1,686.10	11.53	0.62	116,950.97	1,566.68	12,488.67	902.30
sed	JOMP2	75,734.37	1,315.20	11.73	0.44	61,552.53	469.44	14,170.10	1,074.68
sed	JOMP4	50,505.77	1,092.84	11.73	0.44	35,754.93	437.79	14,739.10	1,097.59
sed	GPGPU	20,424.73	1,333.13	467.80	2.20	7,508.57	16.12	12,448.37	1,322.58
space-1	CPU	144,209.27	2,733.97	0.00	0.00	130,875.73	2,677.66	13,333.53	327.63
space-1	JOMP1	80,985.53	1,312.87	13.13	0.34	67,619.57	1,327.43	13,352.83	244.88
space-1	JOMP2	51,000.47	761.16	13.07	0.25	35,512.43	712.74	15,474.97	274.02
space-1	JOMP4	36,331.47	1,763.56	13.17	0.37	21,285.83	526.62	15,032.47	1,454.21
space-1	GPGPU	22,959.10	424.20	467.00	4.86	8,677.07	8.74	13,815.03	424.17
space-2	CPU	140,650.20	2,961.17	0.00	0.00	127,750.53	2,992.23	12,899.67	275.49
space-2	JOMP1	77,375.50	1,296.40	12.97	0.18	64,315.27	1,264.78	13,047.27	253.93
space-2	JOMP2	48,832.40	578.12	13.13	0.34	33,724.50	528.94	15,094.77	294.24
space-2	JOMP4	34,990.77	1,657.58	13.00	0.58	20,272.27	444.72	14,705.50	1,387.08
space-2	GPGPU	21,956.33	388.68	466.50	1.86	8,069.80	11.64	13,420.03	390.20
space-3	CPU	146,322.57	2,154.38	0.00	0.00	133,057.60	2,094.56	13,264.97	232.56
space-3	JOMP1	81,090.77	1,440.19	13.13	0.43	67,842.80	1,410.05	13,234.83	252.39
space-3	JOMP2	51,179.13	956.43	13.10	0.40	35,734.40	895.88	15,431.63	274.58
space-3	JOMP4	36,106.43	1,604.97	13.13	0.50	21,230.63	421.11	14,862.67	1,428.65

Table 13 (continued)

Subject	Config	$\bar{T}_{\text{total}}$	$\sigma_{T_{\text{total}}}$	$\bar{T}_{\text{init}}$	$\sigma_{T_{\text{init}}}$	$\bar{T}_{\text{fitness}}$	$\sigma_{T_{\text{fitness}}}$	$\bar{T}_{\text{remaining}}$	$\sigma_{T_{\text{remaining}}}$
space-3	GPGPU	22,679.07	279.30	466.53	2.01	8,672.27	8.97	13,540.27	277.75
space-4	CPU	143,502.33	3,316.32	0.00	0.00	129,979.30	3,294.07	13,523.03	307.61
space-4	JOMP1	81,009.73	1,379.42	13.07	0.25	67,473.00	1,373.53	13,523.67	273.02
space-4	JOMP2	50,280.77	1,640.55	13.13	0.34	35,278.87	550.71	14,988.77	1,489.76
space-4	JOMP4	36,162.23	1,593.24	12.93	0.25	20,978.10	366.36	15,171.20	1,510.98
space-4	GPGPU	23,215.27	480.33	465.63	1.52	8,675.47	8.98	14,074.17	479.95
replace	CPU	348,104.93	2,187.83	0.00	0.00	281,450.50	1,536.71	66,654.43	1,342.59
replace	JOMP1	469,796.70	1,506.10	13.40	0.55	403,195.30	980.27	66,588.00	1,178.14
replace	JOMP2	292,941.60	3,840.15	13.17	0.37	215,710.27	774.62	77,218.17	3,352.11
replace	JOMP4	254,205.57	3,788.56	13.13	0.34	175,560.03	745.43	78,632.40	3,260.21
replace	GPGPU	85,664.63	735.20	471.57	2.19	18,829.70	31.29	66,363.37	736.89
bash	CPU	2,130,058.33	26,423.09	0.00	0.00	2,099,565.87	26,226.12	30,492.47	296.06
bash	JOMP1	1,363,130.33	25,211.50	53.93	1.57	1,332,820.57	25,252.03	30,255.83	282.71
bash	JOMP2	727,041.77	9,231.70	53.60	1.54	692,904.27	9,237.17	34,083.90	1,158.28
bash	JOMP4	436,764.23	6,370.28	54.33	1.19	402,476.50	6,614.83	34,233.40	1,205.63
bash	GPGPU	92,768.33	298.38	516.80	1.74	62,381.57	12.13	29,869.97	294.00
ibm	CPU	4,605,875.50	460,148.49	0.00	0.00	4,558,144.60	456,561.49	47,730.90	3,707.93
ibm	JOMP1	1,472,382.37	43,731.58	137.50	3.70	1,423,957.80	43,151.02	48,287.07	4,840.65
ibm	JOMP2	805,397.90	19,822.93	136.53	3.79	755,478.63	19,194.49	49,782.73	5,430.14
ibm	JOMP4	496,257.47	15,594.50	135.97	2.96	447,501.90	14,863.69	48,619.60	1,572.93
ibm	GPGPU	187,426.13	6,736.59	645.33	157.36	136,904.93	1,187.38	49,875.87	6,015.35

Table 14 Execution time of two archive algotirhm

Subject	Config	$\bar{T}_{\text{total}}$	σT_{total}	$\bar{T}_{\text{init}}$	σT_{init}	$\bar{T}_{\text{fitness}}$	$\sigma T_{\text{fitness}}$	$\bar{T}_{\text{remaining}}$	$\sigma T_{\text{remaining}}$
printtokens-1	CPU	9,126.60	126.85	0.00	0.00	7,742.80	102.28	1,383.80	45.34
printtokens-1	JOMP1	12,467.17	88.56	5.67	0.47	11,116.57	184.10	1,344.93	132.62
printtokens-1	JOMP2	7,658.47	132.18	5.43	0.50	6,122.23	50.27	1,530.80	155.90
printtokens-1	JOMP4	5,439.40	158.63	5.83	0.37	3,862.43	36.76	1,571.13	163.76
printtokens-1	GPGPU	3,514.80	115.19	471.27	16.17	2,121.33	101.65	922.20	33.19
printtokens-2	CPU	9,397.17	116.54	0.00	0.00	7,966.17	97.71	1,431.00	31.85
printtokens-2	JOMP1	12,594.73	166.41	5.60	0.49	11,217.67	103.84	1,371.47	131.01
printtokens-2	JOMP2	7,799.87	155.56	5.70	0.46	6,231.20	46.99	1,562.97	157.90
printtokens-2	JOMP4	5,536.53	142.51	5.80	0.40	3,914.27	36.16	1,616.47	156.45
printtokens-2	GPGPU	3,619.00	132.69	468.73	2.61	2,173.53	112.02	976.73	32.15
printtokens-3	CPU	9,070.03	107.27	0.00	0.00	7,711.27	100.75	1,358.77	19.02
printtokens-3	JOMP1	12,474.47	147.29	5.70	0.46	11,166.27	108.31	1,302.50	126.92
printtokens-3	JOMP2	7,678.93	128.32	5.67	0.60	6,174.80	36.77	1,498.47	138.76
printtokens-3	JOMP4	5,454.77	158.54	5.33	0.47	3,895.80	31.81	1,553.63	147.48
printtokens-3	GPGPU	3,480.23	91.11	469.60	6.18	2,110.93	78.05	899.70	22.23
printtokens-4	CPU	9,266.03	103.05	0.00	0.00	7,881.93	89.80	1,384.10	29.26
printtokens-4	JOMP1	12,518.37	159.55	5.70	0.46	11,195.47	83.36	1,317.20	147.05
printtokens-4	JOMP2	7,689.80	146.34	5.53	0.50	6,165.00	48.71	1,519.27	170.63
printtokens-4	JOMP4	5,458.00	161.15	5.57	0.50	3,874.77	35.22	1,577.67	162.46
printtokens-4	GPGPU	3,523.97	131.34	470.67	8.37	2,123.60	119.18	929.70	30.61
schedule-1	CPU	6,075.77	84.26	0.00	0.00	4,843.37	63.16	1,232.40	42.59
schedule-1	JOMP1	5,998.63	174.12	3.50	0.50	4,818.23	56.72	1,176.90	162.28
schedule-1	JOMP2	4,110.47	228.08	3.27	0.44	2,776.67	54.51	1,330.53	186.69
schedule-1	JOMP4	3,240.50	239.13	3.53	0.50	1,865.17	49.68	1,371.80	196.83
schedule-1	GPGPU	2,794.13	38.75	475.10	2.68	1,574.63	14.94	744.40	37.45

Table 14 (continued)

Subject	Config	$\bar{T}_{\text{total}}$	$\sigma_{T_{\text{total}}}$	$\bar{T}_{\text{init}}$	$\sigma_{T_{\text{init}}}$	$\bar{T}_{\text{fitness}}$	$\sigma_{T_{\text{fitness}}}$	$\bar{T}_{\text{remaining}}$	$\sigma_{T_{\text{remaining}}}$
schedule-2	CPU	6,127.60	64.40	0.00	0.00	4,890.77	53.02	1,236.83	21.30
schedule-2	JOMP1	6,126.73	167.85	3.37	0.48	4,944.30	46.59	1,179.07	163.06
schedule-2	JOMP2	4,184.70	212.05	3.53	0.50	2,824.23	37.85	1,356.93	195.68
schedule-2	JOMP4	3,282.10	239.99	3.57	0.62	1,883.30	50.56	1,395.23	195.09
schedule-2	GPGPU	2,794.80	17.58	475.23	4.42	1,569.53	13.10	750.03	15.16
schedule-3	CPU	5,937.73	86.90	0.00	0.00	4,756.67	61.36	1,181.07	32.17
schedule-3	JOMP1	5,985.37	180.29	3.33	0.47	4,842.53	67.39	1,139.50	150.55
schedule-3	JOMP2	4,080.90	210.65	3.40	0.49	2,781.77	42.53	1,295.73	177.25
schedule-3	JOMP4	3,182.80	231.91	3.40	0.49	1,859.47	57.61	1,319.93	179.05
schedule-3	GPGPU	2,750.83	34.19	475.00	4.06	1,587.83	16.11	688.00	38.95
schedule-4	CPU	6,041.80	170.24	0.00	0.00	4,818.93	130.01	1,222.87	45.36
schedule-4	JOMP1	6,078.70	177.02	3.50	0.50	4,912.13	59.95	1,163.07	164.25
schedule-4	JOMP2	4,156.27	222.57	3.57	0.50	2,801.57	59.08	1,351.13	197.12
schedule-4	JOMP4	3,247.27	232.23	3.20	0.40	1,881.07	47.90	1,363.00	192.78
schedule-4	GPGPU	2,807.13	37.09	474.17	1.37	1,590.10	23.66	742.87	49.50
printtokens	CPU	116,826.10	624.55	0.00	0.00	106,223.67	599.05	10,602.43	71.58
printtokens	JOMP1	249,776.57	1,050.07	12.03	0.18	239,145.43	1,066.37	10,619.10	95.66
printtokens	JOMP2	142,503.97	679.32	12.13	0.34	130,157.10	739.65	12,334.73	193.63
printtokens	JOMP4	118,927.97	554.48	12.07	0.25	106,493.77	563.52	12,422.13	160.67
printtokens	GPGPU	25,521.53	142.94	471.53	2.93	14,880.73	113.84	10,169.27	60.71
schedule	CPU	62,042.50	865.19	0.00	0.00	55,003.57	828.49	7,038.93	47.23
schedule	JOMP1	126,570.23	554.57	14.63	0.80	119,589.93	564.16	6,965.67	120.73
schedule	JOMP2	73,743.40	367.67	14.73	1.00	65,575.97	380.06	8,152.70	142.48
schedule	JOMP4	60,463.40	441.32	14.67	0.65	52,194.73	499.49	8,254.00	210.47
schedule	GPGPU	15,748.17	124.66	466.57	6.87	8,648.73	81.42	6,632.87	68.09

Table 14 (continued)

Subject	Config	$\bar{T}_{total}$	$\sigma_{T_{total}}$	$\bar{T}_{init}$	$\sigma_{T_{init}}$	$\bar{T}_{fitness}$	$\sigma_{T_{fitness}}$	$\bar{T}_{remaining}$	$\sigma_{T_{remaining}}$
printtokens2	CPU	123,929.63	1,103.11	0.00	0.00	113,296.40	1,078.22	10,633.23	35.64
printtokens2	JOMP1	261,326.87	910.11	12.13	0.34	250,733.73	981.87	10,581.00	198.33
printtokens2	JOMP2	148,466.10	470.87	12.00	0.00	136,103.07	525.71	12,351.03	168.36
printtokens2	JOMP4	123,841.30	433.05	12.07	0.25	111,424.20	436.40	12,405.03	241.67
printtokens2	GPGPU	26,748.17	133.66	471.20	2.01	16,101.30	109.75	10,175.67	53.47
schedule2	CPU	55,375.07	707.72	0.00	0.00	48,138.83	681.86	7,236.23	67.16
schedule2	JOMP1	111,285.10	737.20	14.33	0.47	104,218.50	346.99	7,052.27	618.16
schedule2	JOMP2	65,829.53	287.60	14.90	1.11	57,524.27	354.33	8,290.37	201.67
schedule2	JOMP4	54,595.20	419.74	14.87	1.45	46,196.63	466.64	8,383.70	246.43
schedule2	GPGPU	15,855.27	70.13	464.43	2.63	8,644.97	73.30	6,745.87	41.31
tcas	CPU	20,871.43	90.69	0.00	0.00	16,171.60	71.28	4,699.83	38.20
tcas	JOMP1	31,284.30	245.13	13.93	1.48	26,661.40	308.60	4,608.97	213.03
tcas	JOMP2	20,876.13	183.96	13.37	1.25	15,493.03	253.82	5,369.73	223.50
tcas	JOMP4	16,128.80	224.15	13.27	0.81	10,689.37	139.32	5,426.17	289.61
tcas	GPGPU	9,308.23	166.32	472.33	25.73	4,563.17	142.94	4,272.73	90.77
totinfo	CPU	20,730.80	484.32	0.00	0.00	17,584.63	476.84	3,146.17	28.83
totinfo	JOMP1	30,592.67	208.11	12.93	0.85	27,496.30	182.47	3,083.43	129.72
totinfo	JOMP2	18,946.33	155.43	13.40	1.69	15,360.80	170.58	3,572.13	170.04
totinfo	JOMP4	13,462.13	275.21	12.67	0.54	9,813.60	203.27	3,635.87	171.23
totinfo	GPGPU	6,940.73	38.39	465.30	2.15	3,762.67	33.53	2,712.77	28.69
flex	CPU	71,001.13	549.11	0.00	0.00	70,153.80	541.84	847.33	30.53
flex	JOMP1	41,637.27	1,145.73	13.30	0.46	40,827.47	1,149.14	796.50	133.08
flex	JOMP2	22,405.10	745.27	13.20	0.40	21,483.07	739.98	908.83	158.59
flex	JOMP4	13,882.63	412.73	13.30	0.46	12,958.20	344.10	911.13	159.68
flex	GPGPU	8,171.10	24.95	464.70	1.62	7,274.80	8.56	431.60	25.19

Table 14 (continued)

Subject	Config	$\bar{T}_{total}$	$\sigma_{T_{total}}$	$\bar{T}_{init}$	$\sigma_{T_{init}}$	$\bar{T}_{fitness}$	$\sigma_{T_{fitness}}$	$\bar{T}_{remaining}$	$\sigma_{T_{remaining}}$
gzip	CPU	57,868.43	684.16	0.00	0.00	56,558.87	678.22	1,309.57	29.27
gzip	JOMP1	59,787.40	858.20	12.97	0.75	58,544.00	859.46	1,230.43	151.80
gzip	JOMP2	32,495.43	697.43	12.83	0.37	31,049.13	669.21	1,433.47	160.91
gzip	JOMP4	19,922.97	283.05	12.70	0.46	18,453.80	232.66	1,456.47	172.53
gzip	GPGPU	7,342.70	23.51	465.40	1.99	6,017.30	16.96	860.00	20.26
sed	CPU	101,820.83	2,056.00	0.00	0.00	100,210.43	2,055.16	1,610.40	34.95
sed	JOMP1	119,835.67	1,005.12	11.27	0.44	118,254.57	1,027.73	1,569.83	121.53
sed	JOMP2	63,544.80	665.11	11.23	0.42	61,734.17	690.43	1,799.40	115.92
sed	JOMP4	38,235.60	431.45	11.17	0.37	36,391.20	376.47	1,833.23	124.44
sed	GPGPU	9,386.03	28.80	467.40	1.17	7,753.57	11.73	1,165.07	26.66
space-1	CPU	124,296.60	1,476.27	0.00	0.00	122,756.60	1,459.08	1,540.00	24.09
space-1	JOMP1	69,564.13	1,057.82	13.30	0.46	68,097.53	989.21	1,453.30	166.96
space-1	JOMP2	37,794.93	631.99	13.70	0.46	36,065.37	571.77	1,715.87	133.71
space-1	JOMP4	23,342.97	498.70	13.50	0.50	21,573.17	465.26	1,756.30	121.08
space-1	GPGPU	10,346.67	35.57	467.60	3.10	8,763.73	10.90	1,115.33	39.34
space-2	CPU	121,329.57	1,983.92	0.00	0.00	119,826.43	1,967.27	1,503.13	25.03
space-2	JOMP1	66,364.53	1,008.83	13.70	0.46	64,902.90	923.76	1,447.93	160.68
space-2	JOMP2	35,846.80	494.02	13.40	0.49	34,136.60	466.86	1,696.80	154.29
space-2	JOMP4	21,940.43	360.38	13.37	0.48	20,221.23	257.62	1,705.83	199.13
space-2	GPGPU	9,700.07	25.03	466.83	1.55	8,147.17	9.38	1,086.07	27.75
space-3	CPU	126,114.77	2,653.15	0.00	0.00	124,586.27	2,628.86	1,528.50	34.71
space-3	JOMP1	70,485.43	912.41	13.63	0.48	69,016.53	918.78	1,455.27	153.50
space-3	JOMP2	37,861.03	440.74	13.47	0.50	36,182.00	490.41	1,665.57	185.65
space-3	JOMP4	22,984.63	462.90	13.63	0.48	21,253.93	419.50	1,717.07	208.97
space-3	GPGPU	10,327.40	29.02	467.47	1.89	8,763.63	13.79	1,096.30	29.10

Table 14 (continued)

Subject	Config	$\bar{T}_{total}$	σT_{total}	$\bar{T}_{init}$	σT_{init}	$\bar{T}_{fitness}$	$\sigma T_{fitness}$	$\bar{T}_{remaining}$	$\sigma T_{remaining}$
space-4	CPU	123,416.00	1,596.55	0.00	0.00	121,890.70	1,580.69	1,525.30	25.12
space-4	JOMP1	69,634.33	1,280.05	13.50	0.50	68,162.20	1,222.59	1,458.63	124.06
space-4	JOMP2	37,268.80	670.57	13.67	0.47	35,554.63	620.05	1,700.50	143.71
space-4	JOMP4	22,738.10	273.34	13.40	0.49	21,004.40	227.08	1,720.30	183.28
space-4	GPGPU	10,341.97	31.83	467.63	3.31	8,765.67	9.73	1,108.67	33.49
replace	CPU	197,579.20	4,653.83	0.00	0.00	183,648.37	4,512.88	13,930.83	175.74
replace	JOMP1	420,619.97	906.13	13.27	0.44	406,724.80	864.00	13,881.90	276.66
replace	JOMP2	236,038.03	814.39	13.40	0.49	219,521.77	557.41	16,502.87	632.13
replace	JOMP4	195,827.97	468.77	13.33	0.60	179,560.27	376.03	16,254.37	300.60
replace	GPGPU	36,315.70	322.90	473.40	2.50	22,125.53	179.07	13,716.77	256.86
bash	CPU	1,193,310.07	18,851.34	0.00	0.00	1,188,921.17	18,790.88	4,388.90	69.23
bash	JOMP1	1,354,828.80	20,224.41	53.00	0.97	1,350,286.23	20,221.98	4,489.57	414.13
bash	JOMP2	705,998.83	7,061.49	53.57	0.92	700,673.77	7,103.10	5,271.50	741.65
bash	JOMP4	413,619.80	6,377.16	54.10	1.64	408,339.13	6,412.11	5,226.57	471.50
bash	GPGPU	67,379.67	81.06	518.43	9.40	63,013.40	26.44	3,847.83	71.51
ibm	CPU	2,916,323.63	46,962.47	0.00	0.00	2,914,980.77	46,967.53	1,342.87	35.61
ibm	JOMP1	1,414,298.60	50,912.92	136.37	3.06	1,413,007.20	50,902.83	1,155.03	156.13
ibm	JOMP2	754,894.63	20,118.08	135.70	2.90	753,402.47	20,084.07	1,356.47	147.91
ibm	JOMP4	446,434.20	13,787.74	133.87	3.02	444,931.47	13,775.29	1,368.87	171.03
ibm	GPGPU	139,075.90	84.86	622.10	65.31	137,737.47	84.69	716.33	74.27

References

ATI Stream Computing (2010) OpenCL Programming Guide Rev. 1.05. AMD Corporation

Afzal W, Torkar R, Feldt R (2009) A systematic review of search-based testing for non-functional system properties. Inf Softw Technol 51(6):957–976

Ali S, Briand LC, Hemmati H, Panesar-Walawege RK (2010) A systematic review of the application and empirical investigation of search-based test-case generation. IEEE Trans Softw Eng 36(6):742–762

Asadi F, Antoniol G, Guéhéneuc Y (2010) Concept locations with genetic algorithms: a comparison of four distributed architectures. In: Proceedings of 2nd international symposium on search based software engineering, SSBSE 2010. IEEE Computer Society Press, Benevento, pp 153–162

Black J, Melachrinoudis E, Kaeli D (2004) Bi-criteria models for all-uses test suite reduction. In: Proceedings of the 26th international conference on software engineering, pp 106–115

Boyer M, Tarjan D, Acton ST, Skadron K (2009) 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)

Bull JM, Westhead MD, Kambites ME, Obrzalek J (2000) Towards OpenMP for java. In: Proceedings of the 2nd European workshop on OpenMP, pp 98–105

Chafik O (2009) JavaCL: opensource Java wrapper for OpenCL library (2009). http://code.google.com/p/javacl/. Accessed 6 June 2010

Chau PYK, Tam KY (1997) Factors affecting the adoption of open systems: an exploratory study. MIS Q 21(1):1–20

Chen T, Lau M (1995) Heuristics towards the optimization of the size of a test suite. In: Proceedings of the 3rd international conference on software quality management, vol 2, pp 415–424

Chen TY, Lau MF (1996) Dividing strategies for the optimization of a test suite. Inf Process Lett 60(3):135–141

Clark J, Dolado JJ, Harman M, Hierons RM, Jones B, Lumkin M, Mitchell B, Mancoridis S, Rees K, Roper M, Shepperd M (2003) Reformulating software engineering as a search problem. IEE Proc Softw 150(3):161–175

Cordy JR (2003) Comprehending reality—practical barriers to industrial adoption of software maintenance automation. In: IEEE International Workshop on Program Comprehension, IWPC '03. IEEE Computer Society, pp 196–206

Do H, Elbaum SG, Rothermel G (2005) Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact. Empir Software Eng 10(4):405–435

Durillo J, Nebro A, Alba E (2010) The jmetal framework for multi-objective optimization: design and architecture. In: Proceedings of congress on evolutionary computation 2010. Barcelona, Spain, pp 4138–4325

Durillo JJ, Nebro AJ, Luna F, Dorronsoro B, Alba E (2006) jMetal: a Java framework for developing multi-objective optimization metaheuristics. Tech. Rep. ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos

Ekman M, Warg F, Nilsson J (2005) An in-depth look at computer performance growth. Comput Archit News 33(1):144–147

Engström E, Runeson P, Wikstrand G (2010) An empirical evaluation of regression testing based on fix-cache recommendations. In: Proceedings of the 3rd International Conference on Software Testing verification and validation, ICST 2010. IEEE Computer Society Press, pp 75–78

Engström E, Runeson PP, Skoglund M (2009) A systematic review on regression test selection techniques. Inf Softw Technol 52(1):14–30

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-Completeness. W. H. Freeman and Company, New York, NY

Govindaraju NK, Gray J, Kumar R, Manocha D (2006) Gputerasort: high performance graphics coprocessor sorting for large database management. In: ACM SIGMOD

Harman M (2007) The current state and future of search based software engineering. In: 2007 Future of Software Engineering, FOSE '07. IEEE Computer Society, Washington, DC, pp 342–357

Harman M (2011) Making the case for MORTO: multi-objective regression test optimization. In: 1st international workshop on regression testing (Regression 2011). Berlin, Germany

Harman M, Jones BF (2001) Search based software engineering. Inf Softw Technol 43(14):833–839

Harman M, Krinke J, Ren J, Yoo S (2009) Search based data sensitivity analysis applied to requirement engineering. In: Proceedings of the 11th annual conference on Genetic and Evolutionary Computation, GECCO '09. ACM, Montreal, pp 1681–1688

Harman M, Mansouri A, Zhang Y (2012) Search based software engineering trends, techniques and applications. ACM Comput Surv 45(1):11:1–11:61

Harrold M, Orso A (2008) Retesting software during development and maintenance. In: Frontiers of Software Maintenance, FoSM 2008. IEEE Computer Society Press, pp 99–108

Harrold MJ, Gupta R, Soffa ML (1993) A methodology for controlling the size of a test suite. ACM Trans Softw Eng Methodol 2(3):270–285

Hutchins M, Foster H, Goradia T, Ostrand T (1994) 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, pp 191–200

Kim JM, Porter A (2002) 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, pp 119–129

Kirsopp C, Shepperd M, Hart J (2002) Search heuristics, case-based reasoning and software project effort prediction. In: Proceedings of the genetic and evolutionary computation conference, GECCO 2002. Morgan Kaufmann Publishers, San Francisco, CA, pp 1367–1374

Langdon WB, Banzhaf W (2008) A SIMD interpreter for genetic programming on GPU graphics cards. In: O'Neill M, Vanneschi L, Gustafson S, Esparcia Alcazar AI, De Falco I, Della Cioppa A, Tarantino E (eds) Proceedings of the 11th European conference on Genetic Programming, EuroGP 2008. Lecture notes in computer science, vol 4971. Springer, pp 73–85

Li Z, Harman M, Hierons RM (2007) Search algorithms for regression test case prioritization. IEEE Trans Softw Eng 33(4):225–237

Mahdavi K, Harman M, Hierons RM (2003) A multiple hill climbing approach to software module clustering. In: IEEE international conference on software maintenance. IEEE Computer Society Press, Los Alamitos, CA, pp 315–324

Maia CLB, do Carmo RAF, de Freitas FG, de Campos GAL, de Souza, JT (2009) A multi-objective approach for the regression test case selection problem. In: Proceedings of anais do XLI Simpòsio Brasileiro de Pesquisa Operacional (SBPO 2009), pp 1824–1835

Mitchell BS, Traverso M, Mancoridis S (2001) An architecture for distributing the computation of software clustering algorithms. In: IEEE/IFIP proceedings of the Working Conference on Software Architecture, WICSA '01. IEEE Computer Society, Amsterdam, pp 181–190

Nethercote N, Seward J (2007) Valgrind: a program supervision framework. In: Proceedings of ACM conference on programming language design and implementation. ACM Press, pp 89–100

Offutt J, Pan J, Voas J (1995) Procedures for reducing the size of coverage-based test sets. In: Proceedings of the 12th international conference on testing computer software. ACM Press, pp 111–123

Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, Purcell TJ (2007) A survey of general-purpose computation on graphics hardware. Comput Graph Forum 26(1):80–113

Praditwong K, Yao X (2006) A new multi-objective evolutionary optimisation algorithm: the two-archive algorithm. In: Proceedings of computational intelligence and security, international conference. Lecture notes in computer science, vol 4456, pp 95–104

Premkumar G, Potter M (1995) Adoption of Computer Aided Software Engineering (CASE) technology: an innovation adoption perspective. Database 26(2&3):105–124

Räihä O (2009) A survey on search-based software design. In: Tech. Rep. D-2009-1. Department of Computer Science, University of Tampere

Rothermel G, Harrold MJ, Ostrin J, Hong C (1998) An empirical study of the effects of minimization on the fault detection capabilities of test suites. In: Proceedings of International Conference on Software Maintenance, ICSM 1998. IEEE Computer Society Press, Bethesda, MD, pp 34–43

Rothermel G, Elbaum S, Malishevsky A, Kallakuri P, Davia B (2002a) 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, pp 130–140

Rothermel G, Harrold M, Ronne J, Hong C (2002b) Empirical studies of test suite reduction. Softw Test Verif Reliab 4(2):219–249

Saliu MO, Ruhe G (2007) Bi-objective release planning for evolving software systems. In: Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering, ESEC-FSE 2007. ACM Press, New York, NY, pp 105–114

de Souza JT, Maia CL, de Freitas FG, Coutinho DP (2010) The human competitiveness of search based software engineering. In: Proceedings of 2nd international Symposium on Search Based Software Engineering, SSBSE 2010. IEEE Computer Society Press, Benevento, pp 143–152

Tsutsui S, Fujimoto N (2009) 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, pp 2523–2530

Walcott KR, Soffa ML, Kapfhammer GM, Roos RS (2006) Time aware test suite prioritization. In: Proceedings of the international symposium on software testing and analysis, pp 1–12

Wilson G, Banzhaf W (2009) 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. ACM Press, New York, NY, pp 2531–2538

Wong ML (2009) 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. ACM Press, New York, NY, pp 2515–2522

Wong WE, Horgan JR, London S, Mathur AP (1998) Effect of test set minimization on fault detection effectiveness. Softw Pract Exp 28(4):347–369

Wong WE, Horgan JR, Mathur AP, Pasquini A (1999) Test set size minimization and fault detection effectiveness: a case study in a space application. J Syst Softw 48(2):79–89

Yoo S, Harman M (2007) Pareto efficient multi-objective test case selection. In: Proceedings of international symposium on software testing and analysis. ACM Press, pp 140–150

Yoo S, Harman M (2012a) Regression testing minimisation, selection and prioritisation: a survey. Softw Test Verif Reliab 22(2):67–120

Yoo S, Harman M (2012b) Test data regeneration: generating new test data from existing test data. J Softw Test Verif Reliab 22(3):171–201

Yoo S, Harman M, Ur S (2009) 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, pp 101–110

Yoo S, Harman M, Ur S (2011a) Highly scalable multi-objective test suite minimisation using graphics card. In: LNCS: proceedings of the 3rd international Symposium on Search-Based Software Engineering, SSBSE, vol 6956, pp 219–236

Yoo S, Nilsson R, Harman M (2011b) Faster fault finding at Google using multi objective regression test optimisation. In: 8th European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering, ESEC/FSE '11. Szeged, Hungary. Industry Track

Zhang Y (2011) SBSE repository (2011). http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/. Accessed 14 Feb 2011

Zhang Y, Harman M, Finkelstein A, Mansouri A (2011) Comparing the performance of metaheuristics for the analysis of multi-stakeholder tradeoffs in requirements optimisation. J Inf Softw Technol 53(6):761–773

Shin Yoo is a lecturer of Software Engineering in the Department of Computer Science at University College London. He received his PhD in Software Engineering in 2009 in King's College London. He is widely known for his work on regression testing techniques and Search Based Software Engineering (SBSE). His recent research interests include the use of Genetic Programming to evolve testing and debugging techniques, and the use of Information Theory in software testing.

Mark Harman is professor of Software Engineering in the Department of Computer Science at University College London, where he directs the CREST centre and is Head of Software Systems Engineering. He is widely known for work on source code analysis and testing and was instrumental in founding the field of Search Based Software Engineering (SBSE). SBSE research has rapidly grown over the past five years and now includes over 800 authors, from nearly 300 institutions spread over more than 40 countries. A recent tutorial paper on SBSE can be found here:http://www.cs.ucl.ac.uk/staff/mharman/laser.pdf.

Dr. Shmuel Ur (http://ur-innovation.com) received his PhD in Algorithms Optimization and Combinatorics in 1994 in Carnegie Mellon University (advisers: Michael Trick and Nobel Prize winner Herbert Simon). Shmuel received his Bs.C. and Ms.C. from the Technion in Israel. Shmuel has published academic papers in the fields of hardware testing, artificial intelligence, algorithms, software testing and testing of multi-threaded programs. Shmuel has more than 60 professional publications, more than 30 granted patents and more than 70 filed patents; he has given numerous talks and tutorials at academic conferences and in industrial settings. Shmuel was a research scientist in the IBM research lab in Haifa, Israel for 16 years, where he held the title of IBM Master Inventor. In the last two years Shmuel works as an independent inventor and has sold about 20 ideas, as well as filed quite a few himself. In addition, Shmuel consults start-up companies, helps them in inventing and evaluating their inventions, and advises on their IP strategy. For the past 12 years, at IBM Research and later as an independent consultant, Shmuel has been active in the area of IP education, mainly for computer professionals. He initiated a new workshop entitled Patents and Software Engineering, as well as given multiple workshops on how to generate ideas and what to do with them.