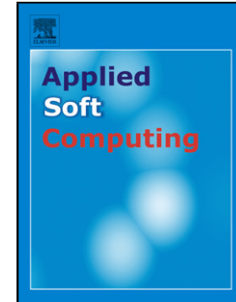
Accepted Manuscript



Title: Tuning compilations by multi-objective optimization: application to Apache web server

Author: Antonio Martínez-Álvarez Sergio Cuenca-Asensi Andrés Ortiz Jorge Calvo-Zaragoza Luis Alberto Vivas Tejuelo

PII: S1568-4946(15)00048-4
DOI: http://dx.doi.org/doi:10.1016/j.asoc.2015.01.029
Reference: ASOC 2739

To appear in: *Applied Soft Computing*

Received date: 28-6-2012
Revised date: 13-12-2014
Accepted date: 7-1-2015

Please cite this article as: Antonio Martínez-Álvarez, Sergio Cuenca-Asensi, Andrés Ortiz, Jorge Calvo-Zaragoza, Luis Alberto Vivas Tejuelo, Tuning compilations by multi-objective optimization: application to Apache web server, *Applied Soft Computing Journal* (2015), http://dx.doi.org/10.1016/j.asoc.2015.01.029

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

*Manuscript
ACCEPTED MANUSCRIPT

Tuning compilations by multi-objective optimization: application to Apache web server

Antonio Martínez-Álvarez[a,*], Sergio Cuenca-Asensi[a], Andrés Ortiz[b], Jorge Calvo-Zaragoza[a], Luis Alberto Vivas Tejuelo[a]

[a]*Department of Computer Technology, University of Alicante, Carretera San Vicente del Raspeig s/n, 03690 Alicante, Spain*
[b]*Department of Communications Engineering, University of Málaga. Campus Universitario de Teatinos s/n, 29071 Málaga, Spain*

Abstract

Modern compilers present a great and ever increasing number of options which can modify the features and behavior of a compiled program. Many of these options are often wasted due to the required comprehensive knowledge about both the underlying architecture and the internal processes of the compiler. In this context, it is usual not having a single design goal but a more complex set of objectives. In addition, the dependencies between different goals is difficult to be *a priori* inferred. This paper proposes a strategy for tuning the compilation of any given application. This is accomplished by using an automatic variation of the compilation options by means of multi-objective optimization and evolutionary computation commanded by the NSGA-II algorithm. This allows finding compilation options that simultaneously optimize different objectives. The advantages of our proposal are illustrated by means of a case study based on the well-known Apache web server. Our strategy has demonstrated an ability to find improvements up to 7.5% and up to 27% in context switches and L2 cache misses, respectively, and also discovers the most important bottlenecks involved in the application performance.

*Corresponding author
Email addresses: amartinez@dtic.ua.es (Antonio Martínez-Álvarez), sergio@dtic.ua.es (Sergio Cuenca-Asensi), aortiz@ic.uma.es (Andrés Ortiz), jcalvo@dtic.ua.es (Jorge Calvo-Zaragoza), lavt@alu.ua.es (Luis Alberto Vivas Tejuelo)

Preprint submitted to Applied Soft Computing *December 13, 2014*

Accepted Manuscript

ACCEPTED MANUSCRIPT

Keywords:
Multi-objective optimization, NSGA-II, compiler, tuning compilations, evolutionary search

1. Introduction

In software solution production, compilers constitute a crucial tool in the software prototype development. Modern compilers expose an ever increasing number of optimization features. However, these options are not fully exploited because that would require a deep knowledge of both the underlying architecture and the target application as well as the compiler usage and internals. The selection of the most convenient set of options to improve a specific goal (e.g. execution time, code size, cache misses, context-switching rate, etc.) represents a task of enormous complexity because there are interdependencies that can not be predicted. Modern most-used compilers, such as GCC [1], CLang [2] or ICC [3], provide a large number of compilation options which can modify the features of the compiled programs.

Many of these compiler options support different modifiers that markedly increase the number of possibilities to compile an application and, consequently, this complicates the selection of the most convenient set of options. Furthermore, certain option combinations can diminish the performance or even change the execution results (for example, code vectorization may relax the floating point accuracy). With thousands of possibilities to consider, estimating the optimal combination to compile a certain code by brute-force is unfeasible in terms of time.

On the other hand, the performance improvement of an application can be characterized by various technical criteria and design constraints that must be satisfied simultaneously and optimized as far as possible. Occasionally, these criteria may conflict and result in a mutual worsening (e.g. performance and power consumption in embedded systems).

Aforementioned reasons lead us to propose a strategy that is able to produce good solutions in an affordable time. In this paper, multi-objective optimization based on a Genetic Algorithm is used to explore the huge solution space. Our proposal runs independently of the underlying compiler under consideration, and it simultaneously optimizes any kind of objective of interest. Furthermore, the process has been accelerated using a parallel scheme based on an island model.

Accepted Manuscript

ACCEPTED MANUSCRIPT

Eventually, the goodness of the proposed approach is shown in a case of study applied to improve the performance of the Apache web server. In this kind of network applications, as they usually involve many processes from the web server, the network stack and the operating system, global compiler optimization options are not enough to optimize. When our method is applied, however, a set of compilation options that simultaneously improve several metrics is obtained.

The rest of the paper is structured as follows: Section 2 includes some related works; Section 3 presents the intrinsic of the compilation process; the proposed system is described in section 4; Section 5 describes the experimental case of study, the obtained results are deeply analyzed in section 6; finally, Section 7 provides the conclusions of this work.

2. Related work

Our solution considers the compiler as a black box so that internal details are transparent to the potential users. This approach was studied before by Pinkers et al. [4] who based their work on orthogonal arrays, inferring the effects of each compiler option in the performance. ACOVEA [5] uses a mono-objective Genetic Algorithm to find the best options for compiling programs with GCC C and C++ compilers. Guy Bashkansky and Yaakov Yaari [6] proposed a framework (ESTO) which obtains suboptimal compilations by using a Genetic Algorithm too. All previous proposals try to optimize a single criterion as a result of the compilation (usually the execution time) and ignore other features that may be equally important in relation to performance.

Only a few works tackle performance improvement as multi-objective optimization problem (MOP), that is, searching the optimal solutions for a set of criteria and taking into account the conflicting interactions among themselves. MOP in combination with GA have been widely studied before [7, 8, 9, 10]. Authors in [11] also adopted a multi-objective evolutionary search for finding the best compiler options.

In this context, the main contributions of our approach are:

- It uses the well-known NSGA-II [12] multi-objective optimization algorithm, to optimize the generated machine code for both, the specific hardware and specific application level objectives.

- It has no dependency on the compiler (compiler agnostic). Moreover, the only need for our optimization technique is a non-interactive appli-

Accepted Manuscript

ACCEPTED MANUSCRIPT

cation whose execution can be operated by means of a set of options at the command line and whose output can be taken from an external program to assess the goodness of a number of objectives of interest.

- It is intended to deal with any kind of objectives of interest. Particularly, we present a case of study to optimize not only the execution of a well known server application, but also the interaction with other possible applications running together (by minimizing L2 cache hits), and the operating system (by minimizing context-switching rate).
- It is parallelized using MPI library on an island model, which reduces the effect of local minima in the search process. In addition, it is implemented in a multi-computer (distributed memory parallel computer) to speed up its execution.

3. Tuning compilations

The compilation process is strongly related to the target architecture. The available compilation options can alter the functioning of the system, as well as the interaction with the operating system, e.g. increasing or decreasing the number of context switches and cache misses, phenomena that can directly affect performance.

Performance of current microprocessors, with their complex pipelines and integrated data and instruction cache, is highly dependent on the compiler and its ability to structure the code for optimal performance. Obtaining the optimal program structure and scheduling is a complex process which is specific to each architecture, leading to large differences in performance depending on the employed optimization techniques. This task is extremely carried out in VLIW (Very Long Instruction Word) and EPIC (Explicitly Parallel Instruction Computing) architectures such as *Itanium* or *Itanium 2*. These microprocessors delegate the instruction scheduling to the compiler in order to reduce the complexity and free up space on the circuit. In these cases, the compiler must statically determine the structure and exploit the parallel architecture to optimize the performance [13].

Besides the above stated, it should be noted that optimization options from modern compilers do not work in an atomic way, i.e. its influence varies depending on other modifiers. Because of this fact, the task of adjusting the compilation in order to take the maximum advantage of the system has an enormous complexity, even with a full knowledge of the process.

Accepted Manuscript

ACCEPTED MANUSCRIPT

Although some compilers include predefined global optimization levels (e.g. `-O1`, `-O2`, `-O3` to speed-up the application, and `-Os` to shrink the generated code, in the case of GCC and CLang compilers), in most applications they are far from being optimal, especially if we are dealing with different contexts (desktop computers, servers, embedded systems, etc.) or different flavor for a given microprocessor architecture (e.g. Intel Pentium D, Intel Core2 Duo or Intel Core i7). In addition, GCC `-O2` global optimization level is used as baseline, providing a reliable optimization level to deploy the software packages in practically any modern Linux distribution. In fact, `-O2` global optimization represents a conservative trade-off between reliability and speed performance, whereas `-O3` decreases the possible microprocessor targets when using aggressive optimization that takes advantage of hardware resources not always available.

Moreover, the use of some optimization techniques can produce adverse effects. For example, function inlining can make a program run faster by avoiding the time cost of routine calls. However, inlining overuse can result in a very large program which directly affects the instruction cache misses and therefore the final performance [14]. Due to these circumstances we propose a strategy which combines a multi-objective optimization scheme for optimizing conflicting goals, with a Genetic Algorithm for speeding up the exploration of the search space.

4. A multi-objective parallel genetic strategy to tune the compilation of applications

In Section 1 the problem of getting the optimal compilation selection has been presented. Considering the number of compilation options which must be taken into account, the problem can not be dealt by exploring all the solution space. Therefore, the proposed strategy uses a genetic algorithm as search engine. Genetic Algorithm (GA) is a stochastic search inspired by natural evolution. It belongs to the group of techniques known as Evolutionary Algorithms (EAs) which are based on the mimicry of evolutionary processes such as natural selection, crossover or mutation. GAs operate with a population of individuals in which each one represents a single solution to the problem. Every individual is codified by its chromosome, which represents a possible solution to the problem. Individuals are randomly initialized and better solutions are obtained by evolving the population through crossover and mutation operators. These individuals are evaluated and selected so

Accepted Manuscript

ACCEPTED MANUSCRIPT

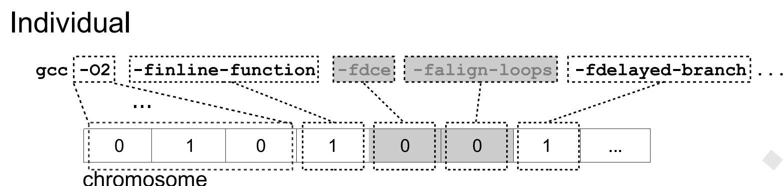


Figure 1: Example of an individual binary codification for GCC compiler. Unset options are represented using a shaded gray background

that only those that represent better solutions can survive. At the end of the process, a set of solutions can be extracted from the surviving population.

For this problem, better solutions will be those that further optimize the target application through the compilation process. Optimization during the compilation consists in the adjustment of certain compiler options in order to improve some features of the program without changing the results, that is, maintaining the correctness. Usually the most common criteria are the reduction of both execution time and code size. Unfortunately, many time optimizations increase the code size and vice versa, so finding a good trade-off between them is not straightforward. But these are not the only conflicting objectives to be taken into account, especially if the scope is not desktop computing. For instance, in embedded systems, there are other factors such as energy consumption, security or fault tolerance. In these situations, where the evaluation of the quality of a solution is unclear, different approaches can be adopted. In this work, a hybrid GA/MOO strategy is adopted.

The following subsections describe the details of the implementation of our algorithm.

4.1. Multi-objective compilation optimization

In our approach, each individual represents a possible compilation with a binary-coded chromosome. We consider two different types of compiler options. The first one defines compiler flags such as `-finline-functions-called-once`, which are encoded using a single gene that enables (1) or disables (0) that option. The second type defines options that take values within a set. An example of this case is the GCC generic optimization levels `-OX` ($X \in \{0, 1, 2, 3, s\}$). These compiler options are encoded using $log_2(n)$ genes, where n is the number of possible values. The binary number indicates which of the possible values is enabled. An example of an individual is illustrated in Fig. 1.

The search evolutionary process consists in five stages (Figure 2). The

Accepted Manuscript

ACCEPTED MANUSCRIPT

population is initialized randomly although initial individuals can be specified to take advantage of previous knowledge. In the evaluation stage, each individual compiles the target application by using the data stored in its genes and evaluates the set of defined criteria. Our implementation uses the MOO algorithm as a way of sorting the population as it will be explained in Section 4.1.1. In the selection stage any surplus individuals are suppressed to control the size of the population. Next, new individuals are created in the crossover stage.

The crossover operator takes two individuals and creates a new one based on them. We implement a uniform crossover, in which the chromosome of every new individual is generated gene by gene choosing randomly (with equal probability) between the values of its parents. Specifically, we cross pairs of randomly chosen individuals until the initial size of the population is doubled. Afterwards, in the mutation stage, three fourths of the population are forced to mutate. The mutation is controlled by a low probability value p_{mut} that some gene of the chromosome changes its value. These stages are repeated until the algorithm computes a maximum number of iterations, specified at the time of launching the search. At the end, the output of the system will be the surviving individuals and the value of its compilation options.

4.1.1. Multi-objective strategy

The main goal of our strategy is to improve the performance of an application through the compilation process. This improvement can be characterized by several metrics of interest that must be optimized simultaneously. Occasionally, these metrics may conflict and result in a mutual worsening (e.g. execution time and code size due to function inlining). For this reason, our approach takes advantage of MOO optimization, which establishes an ordering relation among the set of solutions while taking into account the whole set of criteria at the same time.

In recent years, many different resolution methods have been proposed to solve multi-objective problems. These methods can be classified into two groups: the first are relatively simple algorithms based on Pareto ranking (NSGA [15], NPGA [16], VEGA [17] and MOGA [18]), not currently used, while the second group includes elitist algorithms that emphasize computational efficiency (SPEA [19], SPEA2 [20], NSGA-II [12], MOGLS [21], PESA [22], PESA II [23]). Currently NSGA-II (Non-dominated Sorting Genetic Algorithm II) has been reported to be one of the most successful MOO al-

Accepted Manuscript

ACCEPTED MANUSCRIPT

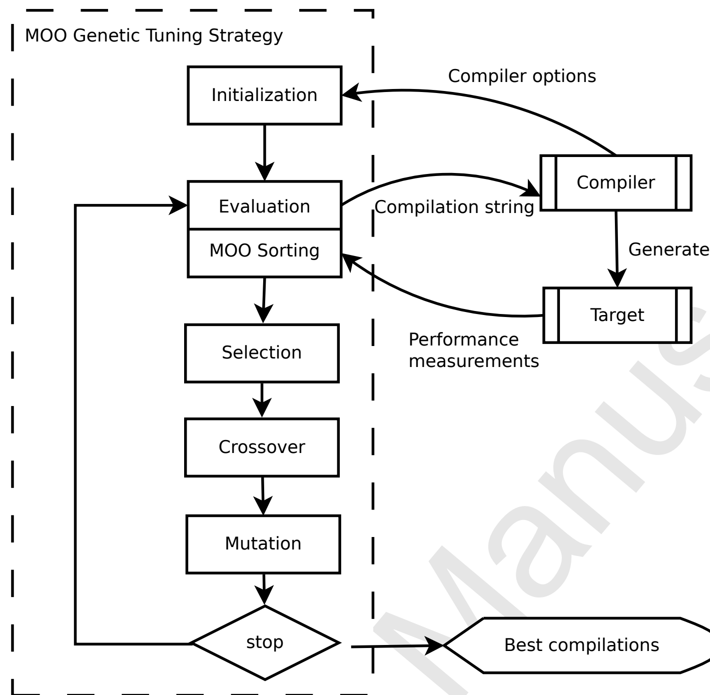


Figure 2: Steps of the MOO genetic tuning strategy

gorithm.

NSGA-II algorithm establishes an order relationship among the individuals of a population mainly based on the concept of non-dominance or Pareto fronts (see Figure 3). It is said that one solution X_i dominates other X_j if the first one is better or equal than the second in every single objective and, at least, strictly better in one of them (i.e. Pareto fronts are defined by those points in which no improvements in one objective are possible without degrading the rest of objectives).

NSGA-II firstly groups individuals in a first front (F_1) that contains all non-dominated individuals, that is the Pareto front. Then, a second front (F_2) is built by selecting all those individuals that are non-dominated in absence of individuals of the first front. This process is repeated iteratively until all individuals are placed in some front.

After the fronts are built, NSGA-II gives another order for the individuals that belong to the same front. To maintain a good spread of solutions, NSGA-II uses the *crowding distance function* (cd) to estimate the diversity value of a solution. Algorithm 1 shows how cd values for a given front F are

Accepted Manuscript

ACCEPTED MANUSCRIPT

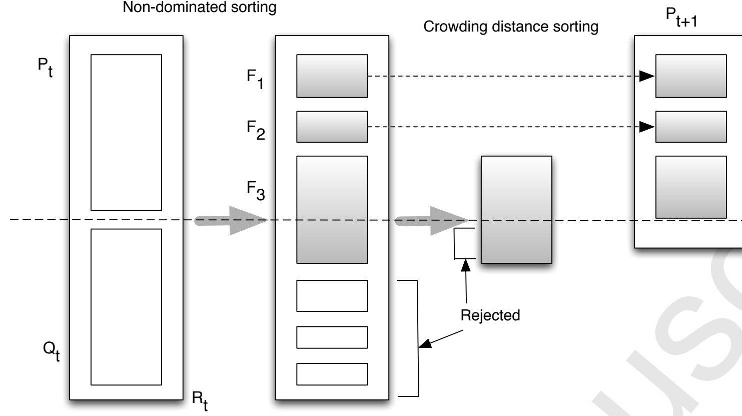


Figure 3: Population sorting and selection using NSGA-II

calculated. Once it is performed, individuals of F are sorted in descending order based on its *cd* value. Therefore those solutions having more diversity are prioritized.

Algorithm 1 Computation of the Crowding Distance

Require: $F = \{I_i\}_{i=1}^{|F|}$: front; M : number of objectives
  for all $F_i \in F$ **do**
    $F_i.cd \leftarrow 0$
  end for
  for all $m \leq M$ **do** # Loop over each objective
    $F \leftarrow \mathbf{sort}(F, m)$ # Sort using the single objective m
    $v_m^{max} \leftarrow \mathbf{max}(F, m)$
    $v_m^{min} \leftarrow \mathbf{min}(F, m)$
    $F_1.cd \leftarrow F_{|F|}.cd \leftarrow \infty$ # Boundary points
    for all $i = 2$ **to** $|T| - 1$ **do**
      $F_i.cd \leftarrow F_i.cd + \frac{F_{i+1}.m - F_{i-1}.m}{v_m^{max} - v_m^{min}}$
    end for
  end for

Summarizing, after the execution of the NSGA-II algorithm, the population is first sorted by non-dominated fronts and then, by the crowding distance (Fig. 3).

Accepted Manuscript

ACCEPTED MANUSCRIPT

4.2. Parallelizing the optimization algorithm

The evaluation stage takes up most of the time in our scenario due to the fact that it requires the execution of the target application or, at the very least, its compilation. Fortunately, since each individual can be evaluated independently this problem presents a great level of parallelism. Nevertheless, in order to make the most of the parallelization, the island model has also been implemented. This model considers several different populations and simultaneously iterate over all of them. The islands exchange individuals of their population in order to improve the search engine by helping to keep the diversity of solutions. This can also help to avoid as far as possible the likelihood to come to a standstill because of local maxima, as reported on [24, 25].

In our implementation, after a certain number of iterations our islands exchange individuals within F_1 (given by the MOO order relationship as indicated in Section 4.1.1). This migration stage does not modify the inner working of the system explained previously in section 4.1 because it is included as a new step at the end of the GA main loop. Therefore, it has no interference in other stages.

As many islands may be considered, this process could cause execution and network overload. To alleviate this situation, instead of exchanging individuals among every pair of islands, only some combinations are taken into account in every execution of the migration stage. Hence, at the migration stage, pairs of islands are chosen randomly where one of them acts as sender and the other as receiver. The process that coordinates the island model is in charge of doing this choices and notifying its role to every island (*sender*, *receiver* or *not involved*). The sender island sends its individual to the receiver island which includes them into its population as conventional individuals. The following evaluation and sorting stages of the GA (see Fig. 2) will determine the goodness of the migrated individuals in the receiving population as it is done with the rest of the population.

Within the island model, our proposal implements a parallelization using the Message Passing Library (MPI) [26] and a master-slave programming structure (see Fig. 4): n processes are launched as islands (executing its own genetic search), taking the *master* role; these processes are provided each one with m *slave* processes. The slaves, also called *fitness* processes, are in charge of receiving a solution, using the associated compilation to generate the application and measure the considered metrics of interest (fitness of

ACCEPTED MANUSCRIPT

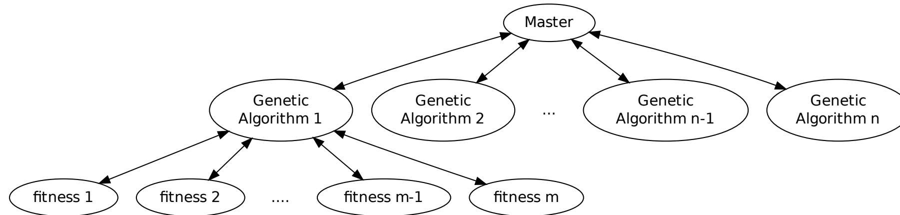


Figure 4: Process tree created in our proposal. There are N populations, each of which have a set of fitness processes.

the individual). Thus, simultaneous evaluation of solutions is carried out. Afterwards, fitness processes return the set of results to its island process.

5. Case of study: optimization of the Apache web server

The experimental part of this work aims to show the effectiveness of the proposed approach to find a set of compilation options that maximize a set of predefined criteria in an interesting case of study. To this end, the proposed strategy is applied in an effort to improve the performance of the Apache web server running on a Linux machine.

The exponential demands for high performance web servers is a trend which is gaining importance in the world of information and business-oriented services [27]. Modern web servers devote most of the execution time to the network operating system stack. Therefore, the network stack has a great influence on the global performance. In these applications, the number of cache misses is usually high due to the fact that the Linux TCP/IP stack does not fit in usual sized L2 cache. Furthermore, if pages with random data (which can avoid the microprocessor predictors) are being used, the L2 cache misses can increase as well. As latency of the DRAM is usually over a hundred clock cycles, these cache misses greatly reduce the throughput of the processor. Therefore, the optimization of Apache web server involves the following metrics of interest: number of operating system context switches (obtained from `/proc/stat`), L2 cache misses, web server throughput and mean time per request. All mentioned criteria must be taken into account to measure the overall performance.

The evaluation of the web server performance was done through the use of Apache Benchmark (AB) tool [28]. It allows to do load tests with different

Accepted Manuscript

ACCEPTED MANUSCRIPT

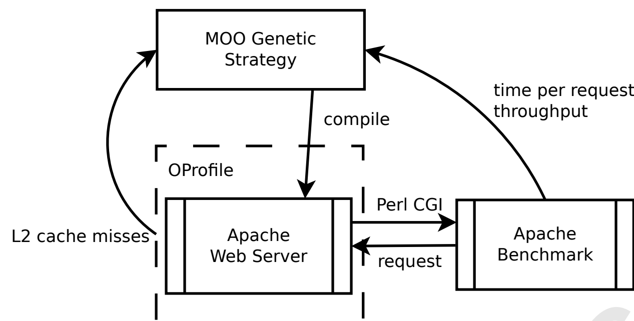


Figure 5: Scenario of the Apache case of study

configurations and it provides a range of performance indicators, such as the number of requests processed per second or the number of failed requests. On the other hand, OProfile [29] was used to do cache performance measurements. OProfile is a fine-grained code profiling tool which consists of a kernel driver, a daemon and many reporting tools. By means of sampling, it collects data from the performance counter registers within defined time intervals. Unlike other tools, OProfile can profile the whole system without the need to modify the target application. Figure 5 represents the scenario of the case of study.

To take into account the worst-case scenario, avoiding the CPU predictors is mandatory [30]. Thus, the requested page was established as a Perl CGI script that generates random real numbers printed on a single line. This test also allows us to have more precise measurements since it minimizes I/O usage. Test have been done with page lengths of 2 KiB and 32 KiB. The overall optimization process over 145 generations with 34 individuals took about 6 days.

The experimental setup is composed of Intel Core 2 Duo 2.3GHz machines with 4 GiB of RAM and a Realtek RTL8169 based gigabit network interface controller (NIC), running Debian Linux with the 3.2.0-1-amd64 kernel and GCC 4.6.2. The NIC maximum transmitted unit (MTU) was configured to the maximum allowed value of 7000 to optimize the bandwidth. Apache version 2.2.21 was used in our tests.

The parameters of the genetic operation are selected as follows: population size 32, mutation probability (P_{mut}) 0.05 and number of iterations 200.

ACCEPTED MANUSCRIPT

5.1. *Experimental results*

The results are expressed with a set of points closest to the origin of coordinates in both axes. GCC global optimization levels (`-O1`, `-O2`, `-O3`, `-Os`) are within this set and represent a baseline to compare with the improvements achieved by our method. It is worth noting that `-O2` and `-O3` global compilation options do not provide 100 % compatible code for each microprocessor flavor from a given architecture, as they generate optimized code that may take advantage of specific hardware resources not always available in each microprocessor version for the same architecture.

The data are shown in graphs facing pairs of criteria. The individuals belonging to the Pareto front are identified by means of blue dots.

5.1.1. *2 KiB page*

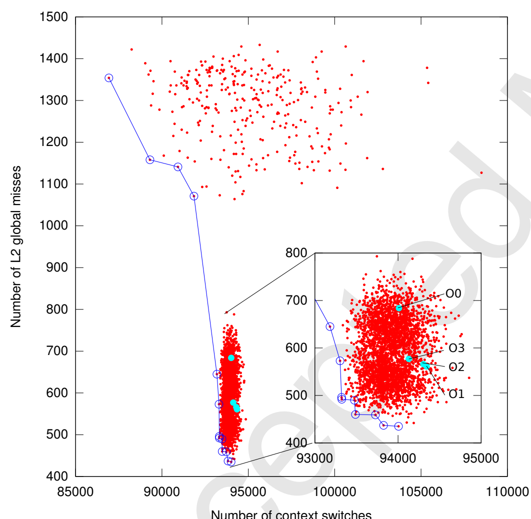


Figure 6: L2 cache misses against context switches

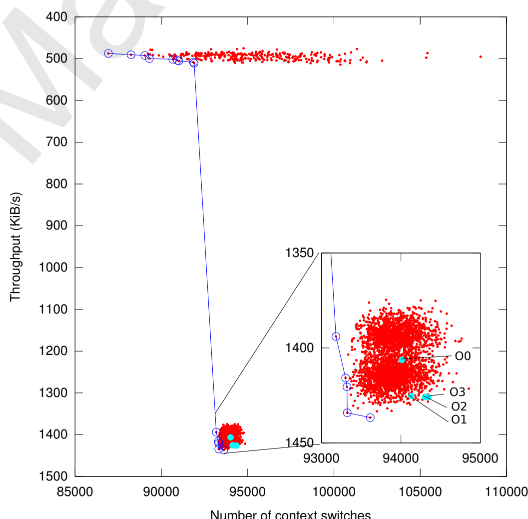


Figure 7: Throughput against context switches

There are two different groups consistently throughout all results. Figure 6 shows a point cloud with the set of solutions obtained throughout the execution based on L2 cache misses and the number of context switches. The solutions belonging to the Pareto front range from individuals with a low number of context switches and a high amount of L2 cache misses to more balanced individuals. The latter shows a significant reduction of cache misses compared to the predefined optimization levels.

Accepted Manuscript

ACCEPTED MANUSCRIPT

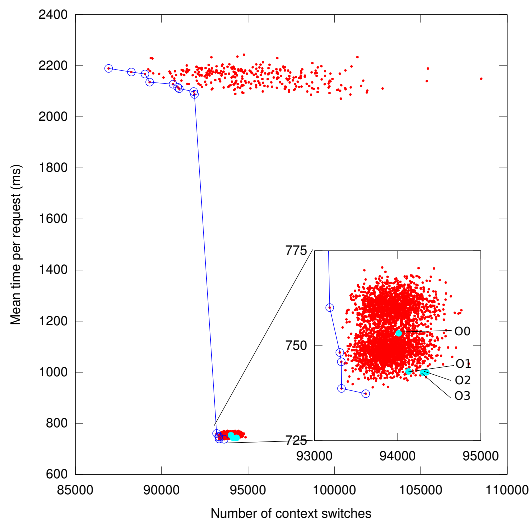


Figure 8: Time per request (mean) against context switches

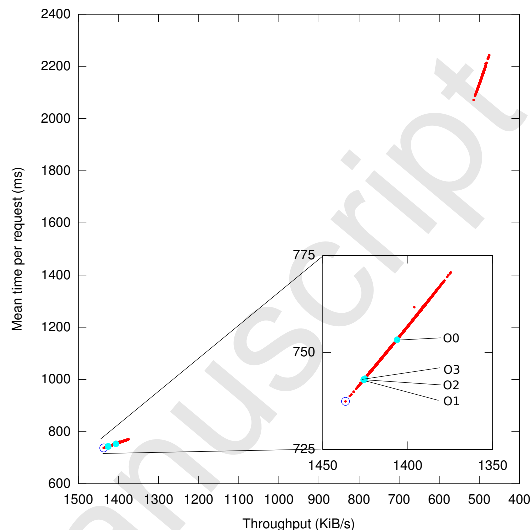


Figure 9: Time per request (mean) against throughput

Figure 7 shows the solutions in terms of throughput (in KiB/s) and context switches. Similarly to the previous graph (Figure 6) there are two different groups, more so in this case since the groups are more compact. Figure 8 represents the time per request and the number of context switches. The behavior is similar in both cases and express the inverse proportionality between those two criteria. There are differences however, because of the time per request being a mean value, so that the relationship is not necessarily linear (Figure 9).

x-axis in Figure 10 represents the number of L2 cache misses while the *y-axis* represents the throughput. A relation can be seen between these two criteria: individuals with the highest number of cache misses get a much lower throughput. The predefined optimization levels, nevertheless, produce a good throughput despite having a worse result in cache misses. The solutions belonging to the Pareto front have a marginally higher throughput and are significantly better in terms of cache misses. The situation is similar when comparing the mean time per request and the L2 cache misses (Figure 11).

5.1.2. 32 KiB page

Figures 12, 13, 14, 15, 16 and 17 show similar patterns and trends, with two clearly differentiated groups and little-differentiated Pareto fronts. Again, the default optimization levels from GCC are within the point cloud

Accepted Manuscript

ACCEPTED MANUSCRIPT

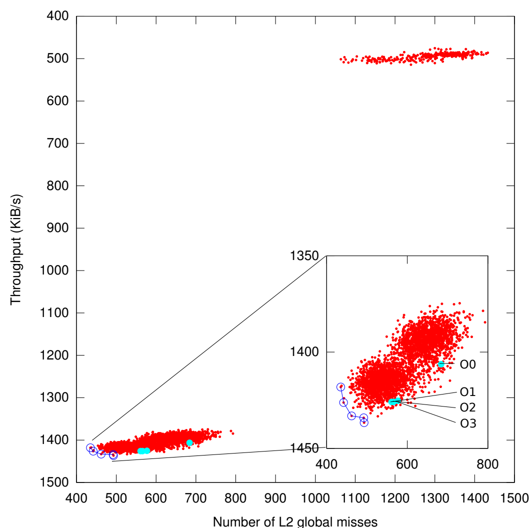


Figure 10: Throughput against L2 cache misses

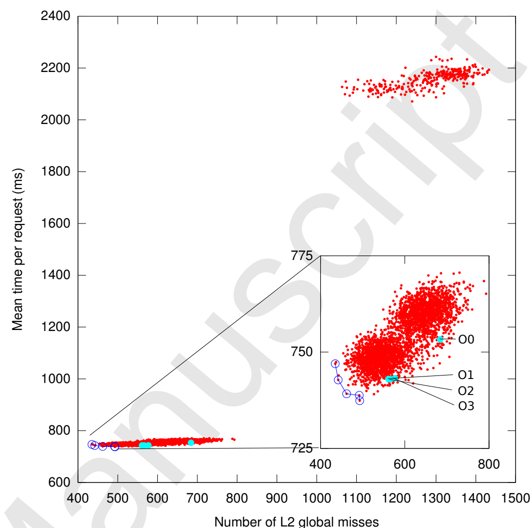


Figure 11: Time per request (mean) against L2 cache misses

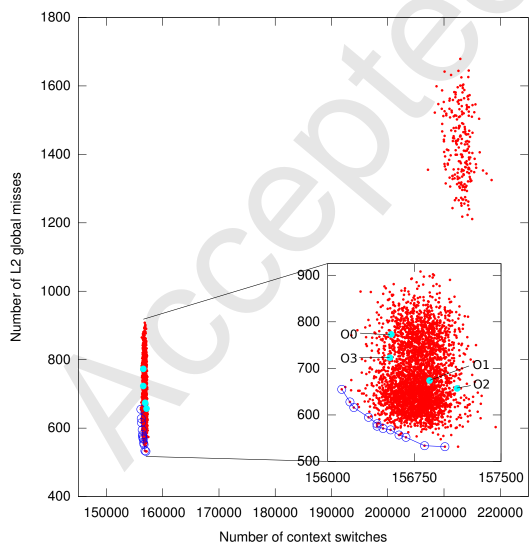


Figure 12: L2 cache misses against context switches

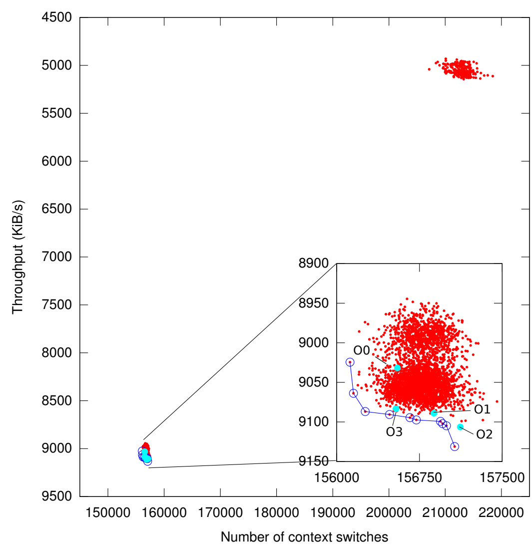


Figure 13: Throughput against context switches

Accepted Manuscript

ACCEPTED MANUSCRIPT

of individuals that yield better results.

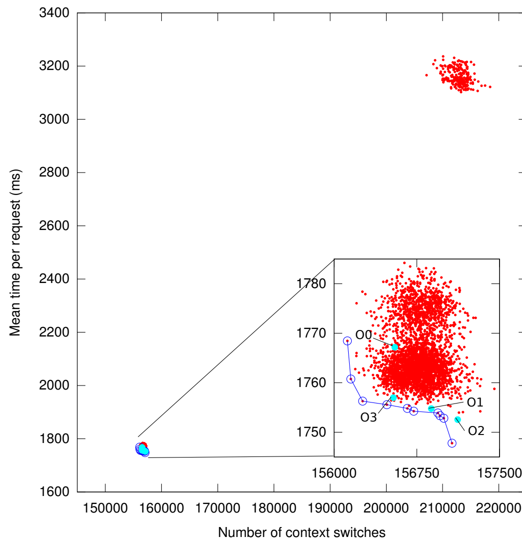


Figure 14: Time per request (mean) against context switches

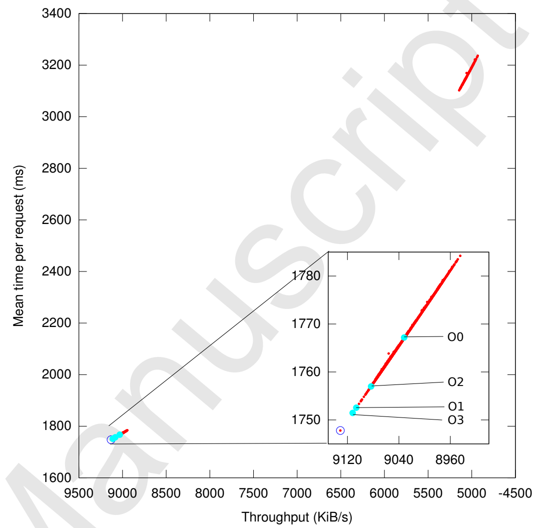


Figure 15: Time per request (mean) against throughput

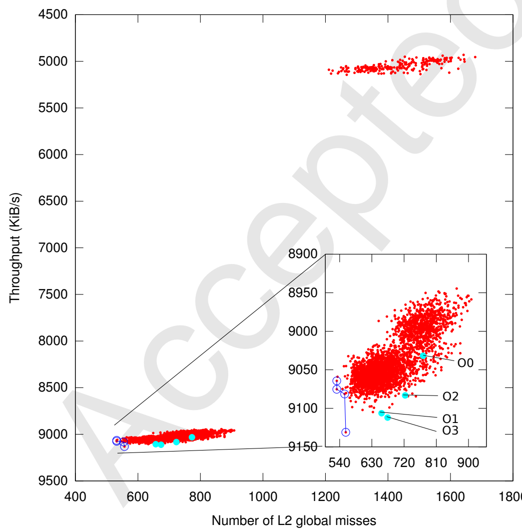


Figure 16: Throughput against L2 cache misses

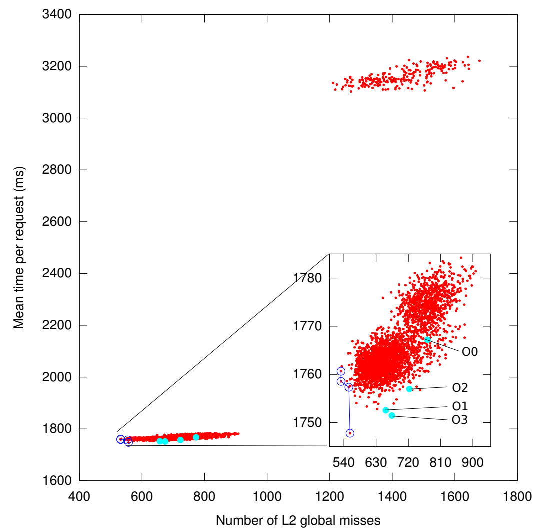


Figure 17: Time per request (mean) against L2 cache misses

Accepted Manuscript

Finally, if the optimization interest is on every single criterion separately, a simplification of the multiobjective problem to four mono-objective prob-

ACCEPTED MANUSCRIPT

lems, can be done. In this context Table 1 shows the maximum % of improvement for the best case in each single criterion across every individual from all generations and for 2KiB and 32KiB pages.

	CS	L2	TPUT	MTPR
2 KiB	8,5	30,1	0,8	0,7
32 KiB	0,3	29,8	0,5	0,5

Table 1: Maximum % of improvement for the best case in each single criterion (CS: Context switching, L2: L2 cache misses, TPUT: Throughput, MTPR: Mean Time per Request) across every individual from all generations for 2KiB and 32KiB pages.

6. Discussion

In this case study, Apache web server has been optimized to improve its performance by following simultaneously diverse criteria of interest. Specifically, the following four criteria have been selected in the conducted experiments:

1. Minimize total number of context switches across all CPUs: criterion strongly linked to the interaction of the application under study and the operating system scheduler.
2. Minimize L2 cache misses across all CPUs: which represent a very important criterion when having applications with a high network processing workload. In this sense, the TCP/IP stack of an operating system is a well-known bottleneck in the communication path which relies in the Linux kernel [31] [32] [33]. Therefore, tuning the compilation of a network application to avoid unnecessary overload on L2 cache is a must.
3. Maximize web server throughput: criterion strongly related with any web server.
4. Minimize mean time per request: as the previous criterion, this one is also strongly related with any web server. In addition, in this experiment a theoretical linear relationship between these two criteria ($TimePerRequest \propto Throughput$) must be revealed and tested to ensure the correctness of the experimental setup. With this purpose, this criterion has been introduced.

Accepted Manuscript

ACCEPTED MANUSCRIPT

Analyzing the results of every Pareto-front figure within this paper (with served pages lengths of both 2 and 32KiB), we note that GCC predefined optimization levels do not provide a significant performance improvement by themselves. Moreover, their graphical location show a distribution of always dominated solutions very far from the optimal Pareto Front. Focusing our attention to the average improvement of the four criteria by taking into account the best candidates in every generation, we achieved an improvement of 7.5% and 8.8% when serving 2 KiB and 32 KiB pages respectively. It is important to highlight the dependence of the overall improvement level on the served page length, showing that changes in the processed workload length have a mayor influence on cache hits and context switches.

It is interesting to remark the spatial distribution of red points (or dominated solutions) across all Pareto-front figures where we can observe some aggregations of points always located next to the Pareto front. These points depict a solution space where mutations have little to no effect on the final solution. More than one aggregation of this type can be observed, leading us to think that an important mutation has occurred. In addition, in some graphs, the cloud of points resembles a straight line with slope different to 0 or ∞, which clearly indicates a linear dependency between the facing criteria. In this case, we can omit one criterion (as one is a function of the other), nevertheless as pointed above we have left them with the purpose of testing and ensuring the correctness of experimentation. This behavior can be observed in Figures 9 and 15, where mean time per request as a function of throughput is represented. In these figures, as pointed at the beginning of this section, the proportional relation $TimePerRequest \propto Throughput$ is revealed. As shown in Table 1, while 30% of improvement in L2 cache misses is achieved for the 2KiB case, the number of context switches (CS) is reduced in 8,5%. These two objectives contribute to release CPU cycles which can be taken by the application. Regarding the 32 KiB case, while L2 cache misses reduction is similar to the 2KiB case, context switching reduction drops to 0,3%. As larger pages tend to produce a lower number of context switches, the optimization is less evident in this case.

Figure 18 shows the improvement provided by different compilation options corresponding to non-dominated solutions (MOO1, MOO2, MOO3 and MOO4) with respect to `-O2` compilation. Moreover, improvements provided by `-O0`, `-O1` and `-O3` are provided for comparison. Graphs in Figure 18 show that optimized compilations using the non-dominated solutions provide clear improvements over the `-O2` compilation. Specifically, improvements in con-

Accepted Manuscript

ACCEPTED MANUSCRIPT

text switches are specially relevant as they determine the performance of the network subsystem when small packets are used [33, 34, 32]. Thus, improvements in context switches are more noticeable for small packets, as expected [34, 35, 32]. Additionally, cache misses are reduced for both 2KiB and 32KiB case. Again, compilations generated by non-dominated solutions clearly outperform -O0, -O1, -O2, and -O3 compilations. Regarding throughput and mean time per request improvements, these are less noticeable as maximum throughput is usually limited by packet size [32]. However, improvements in context switches and L2 cache misses decidedly contribute to release CPU cycles that remains available to other processess [34].

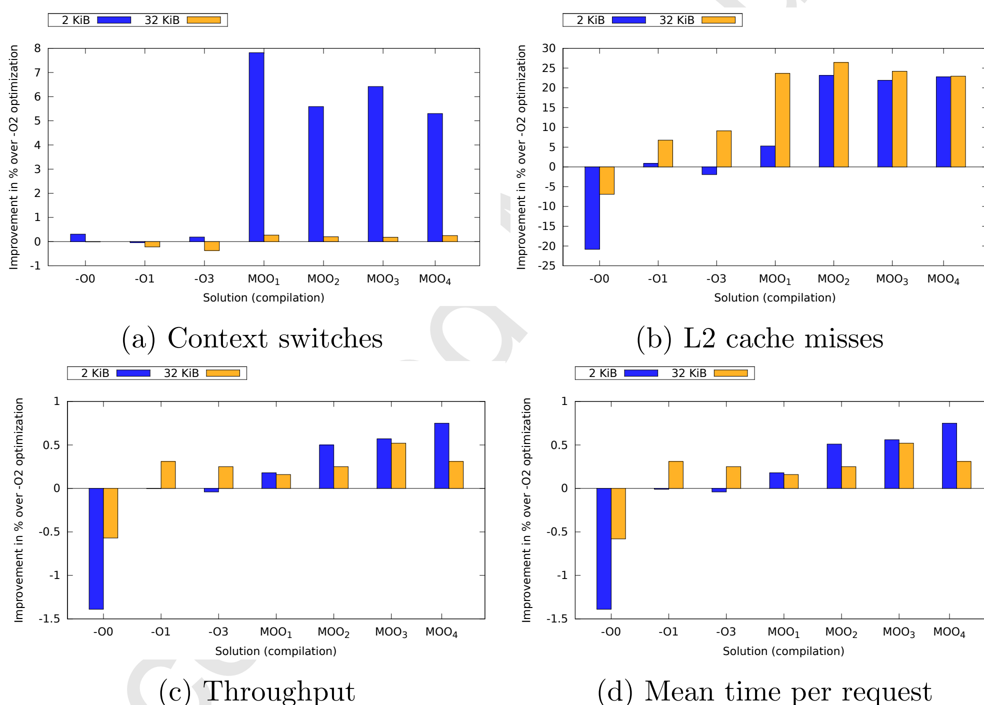


Figure 18: Improvement in (%) achieved by the non-dominated solutions with respect to -O2 in the considered criteria.

7. Conclusions

In this paper we present an optimization tool which figures out the compiler options that aims to maximize the performance of a specific application. As multiple criteria are required in the optimization process, this tool has

ACCEPTED MANUSCRIPT

been implemented using a multi-objective Genetic Algorithm that can work in parallel in a multi-computer environment, speeding up the overall process. It has been observed that the presented technique performs well when dealing with optimization problems with multiple criteria satisfaction, by means of the implemented sorting and conservation of the best solutions strategy based on the algorithm NSGA-II. In addition, the presented tool may reveal the relationship among objectives. This is an important feature as the results can be used for further software and hardware improvements.

In our case of study, focused on optimizing the operation of Apache, we found a limited room for improvements with respect to the default compilation in each criterion separately. However, we show improvements in the cache operation and a relationship between cache hits and web server performance is figured out. This way, our strategy has demonstrated an ability to find improvements up to 7.5% and up to 27% in context switches and L2 cache misses, respectively, contributing to release CPU cycles that remains available to other processes.

As a future direction we will attempt to infer the incidence of compilation options to each criterion. In this case, it would be interesting to know which option or combination of options makes individuals belong to a better or worse front as seen in the graphs. Another interesting field of study would be to consider other applications that do not have such a high dependency on network resources in order to compare the results. This would also allow us to make a study of the incidence of compilation options depending on the application type, whose results could be used in the strategy as a-priori knowledge. In fact, the presented technique can be used with every application and platform, constituting a valuable tool to optimize the performance of a specific application. Moreover, the optimization process also discovers the most important bottlenecks involved in the application performance through the relationships between objectives, which can be used to improve the software and to optimally scaling the server hardware.

References

[1] GCC, GNU Compiler Collection, `http://gcc.gnu.org`, 1987.

[2] CLang, A C language family frontend for LLVM, `http://clang.llvm.org/`, 2011.

Accepted Manuscript

ACCEPTED MANUSCRIPT

[3] ICC, Intel C++ Compiler, http://software.intel.com/en-us/intel-compilers/, 2011.

[4] R. P. J. Pinkers, P. M. W. Knijnenburg, M. Haneda, H. A. G. Wijshoff, Statistical Selection of Compiler Options, in: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS '04, IEEE Computer Society, Washington, DC, USA, ISBN 0-7695-2251-3, 494–501, URL http://dl.acm.org/citation.cfm?id=1032659.1034235, 2004.

[5] S. R. Ladd, ACOVEA, http://kidsvid.altec.org Last accessed: 30/03/2012, 2007.

[6] Black box approach for selecting optimization options using budget-limited genetic algorithms, SMART '07, 2007.

[7] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms – A comparative case study, in: A. Eiben, T. Bäck, M. Schoenauer, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature — PPSN V, vol. 1498 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, ISBN 978-3-540-65078-2, 292–301, URL http://dx.doi.org/10.1007/BFb0056872, 1998.

[8] C. Fonseca, P. Fleming, Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, in: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 416–423, 1993.

[9] F. Pettersson, N. Chakraborti, H. Saxén, A genetic algorithms based multi-objective neural net applied to noisy blast furnace data, Applied Soft Computing 7 (1) (2007) 387–397, ISSN 1568-4946, URL http://dx.doi.org/10.1016/j.asoc.2005.09.001.

[10] G. Ascia, V. Catania, A. G. D. Nuovo, M. Palesi, D. Patti, Performance evaluation of efficient multi-objective evolutionary algorithms for design space exploration of embedded computer systems, Applied Soft Computing 11 (1) (2011) 382–398, ISSN 1568-4946, URL http://www.sciencedirect.com/science/article/pii/S1568494609002427.

Accepted Manuscript

ACCEPTED MANUSCRIPT

[11] K. Hoste, L. Eeckhout, Cole: Compiler Optimization Level Exploration, in: Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO '08, ACM, New York, NY, USA, ISBN 978-1-59593-978-4, 165–174, URL http://doi.acm.org/10.1145/1356058.1356080, 2008.

[12] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197, ISSN 1089-778X.

[13] C. Dulong, R. Krishnaiyer, D. Kulkarni, An Overview of the Intel IA-64 Compiler, 1999.

[14] F. P. Miller, A. F. Vandome, J. McBrewster, Inline Expansion: Compiler Optimization, Called Party, Branch (computer science), Algorithmic Efficiency, Return Statement, CPU Cache, Constant (programming), Compiler, Copy and Paste Programming., Alpha Press, ISBN 6130631146, 9786130631147, 2010.

[15] N. Srinivas, K. Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, Evol. Comput. 2 (3) (1994) 221–248, ISSN 1063-6560, URL http://dx.doi.org/10.1162/evco.1994.2.3.221.

[16] J. Horn, N. Nafpliotis, D. Goldberg, A niched Pareto genetic algorithm for multiobjective optimization, in: Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, 82–87 vol.1, 1994.

[17] J. D. Schaffer, Multiple Objective Optimization with Vector Evaluated Genetic Algorithms, in: Proceedings of the 1st International Conference on Genetic Algorithms, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, ISBN 0-8058-0426-9, 93–100, URL http://dl.acm.org/citation.cfm?id=645511.657079, 1985.

[18] C. Fonseca, P. Fleming, Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, in: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 416–423, 1993.

Accepted Manuscript

ACCEPTED MANUSCRIPT

[19] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271, ISSN 1089-778X.

[20] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, in: K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, T. Fogarty (Eds.), Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, International Center for Numerical Methods in Engineering, Athens, Greece, 95–100, 2001.

[21] A. Jaszkiewicz, On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment, Evolutionary Computation, IEEE Transactions on 6 (4) (2002) 402–412, ISSN 1089-778X.

[22] D. Corne, J. D. Knowles, M. J. Oates, The Pareto Envelope-Based Selection Algorithm for Multi-objective Optimisation, in: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature, PPSN VI, Springer-Verlag, London, UK, UK, ISBN 3-540-41056-2, 839–848, URL http://dl.acm.org/citation.cfm?id=645825.669102, 2000.

[23] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates, M. J, PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001, Morgan Kaufmann Publishers, 283–290, 2001.

[24] D. Van Veldhuizen, J. Zydallis, G. Lamont, Considerations in engineering parallel multiobjective evolutionary algorithms, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 144–173, ISSN 1089-778X.

[25] F. de Toro, J. Ortega, J. Fernandez, A. Diaz, PSFGA: a parallel genetic algorithm for multiobjective optimization, in: Proceedings of 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, 384–391, 2002.

[26] MPI Forum, Message Passing Interface (MPI) Forum Home Page, http://www.mpi-forum.org/, Dec. 2009.

ACCEPTED MANUSCRIPT

[27] S. Sharifian, S. A. Motamedi, M. K. Akbari, A predictive and probabilistic load-balancing algorithm for cluster-based web servers, Applied Soft Computing 11 (1) (2011) 970–981, ISSN 1568-4946, URL http://www.sciencedirect.com/science/article/pii/S1568494610000220.

[28] Apache HTTP server benchmarking tool, The Apache Software Foundation, http://httpd.apache.org/docs/2.0/programs/ab.html, 2011.

[29] OProfile, A System Profiler for Linux, http://oprofile.sourceforge.net/, 2011.

[30] Intel 64 and IA-32 Architectures Optimization Reference Manual, Intel, URL http://www.intel.com/content/dam/doc/manual/64-ia-32-architectures-optimizatic 2011.

[31] W. Wu, M. Crawford, M. Bowden, The Performance Analysis of Linux Networking - Packet Receiving, Computer Communications 30 (5) (2007) 1044–1057, ISSN 0140-3664, URL http://dx.doi.org/10.1016/j.comcom.2006.11.001.

[32] N. L. Binkert, L. R. Hsu, A. G. Saidi, R. G. Dreslinski, A. L. Schultz, S. K. Reinhardt, Analyzing NIC Overheads in Network-Intensive Workloads, in: In 8th Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW), Feb 2005, 1–23, 2004.

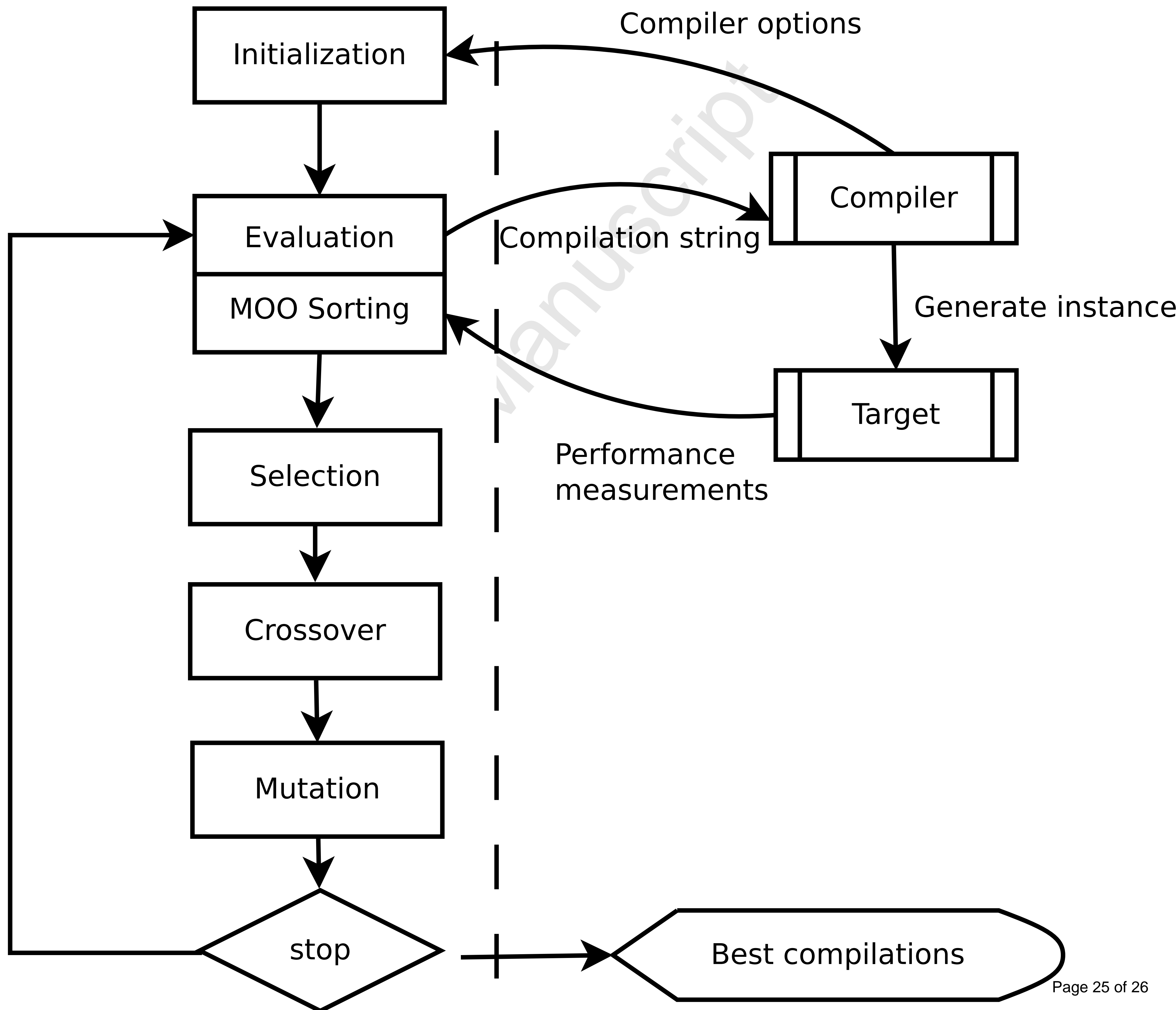
[33] A. Ortiz, J. Ortega, A. F. Díaz, A. Prieto, Network Interfaces for Programmable NICs and Multicore Platforms, Computer Networks 54 (3) (2010) 357–376, ISSN 1389-1286, URL http://dx.doi.org/10.1016/j.comnet.2009.09.011.

[34] A. Ortiz, J. Ortega, A. Díaz, A. Prieto, Affinity-Based Network Interfaces for Efficient Communication on Multicore Architectures, Journal of Computer Science and Technology 28 (3) (2013) 508–524, ISSN 1000-9000, URL http://dx.doi.org/10.1007/s11390-013-1352-2.

[35] A. Ortiz, J. Ortega, A. Díaz, M. Anguita, Leveraging bandwidth improvements to web servers through enhanced network interfaces, The Journal of Supercomputing 65 (3) (2013) 1020–1036, ISSN 0920-8542, URL http://dx.doi.org/10.1007/s11227-012-0841-3.

Accepted Manuscript

*Graphical abstract (for review)
D MANUSCRIPT



ACCEPTED MANUSCRIPT

- A strategy and tool for tuning compilations by multiobjetive optimization is proposed.
- Multiple criteria satisfaction is based on NSGA II algorithm.
- The tool has been tested using a case of study based on Apache web server.
- Overall Apache improvement achieved up to 8.5% assuming 4 criteria of interest.
- 30,1% of improvement when a single criterion (minimize L2 cache misses) is selected.

Accepted Manuscript