The Register Allocation and Instruction Scheduling Challenge

João F. N. Carvalho
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, M.G., Brasil
joaofnc@dcc.ufmg.br

Bruno L. Sousa
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, M.G., Brasil
bruno.luan.sousa@dcc.ufmg.br

Marcus R. Araújo
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, M.G., Brasil
maroar@dcc.ufmg.br

Mariza A. S. Bigonha
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, M.G., Brasil
mariza@dcc.ufmg.br

ABSTRACT

Both register allocation and instruction scheduling are old and open issues in Computer Science, despite the efforts already made to address them separate or jointly. Register allocation may be seen as having two parts: *allocation*, which decides which values should be in registers, and *assignment*, which assigns a specific register to each value. Instruction scheduling aims at identifying and moving the instructions in the code, changing their original execution sequence, so that they may run in parallel. Register allocation and instruction scheduling attempt to minimize the execution time of the program, however, they are interdependent and are involved in a prioritization problem. This paper presents a *Systematic Literature Review* (SLR) related to this problem. From a total of 542 primary studies initially obtained on six databases, 25 studies closely related to this research theme were identified, 12 of them published between 2000 and October 2016. These studies were analyzed to answer the research questions proposed in this SLR, producing useful informations about this theme and about the approaches that, traditionally, have been used to solving this problem. An important finding of this research is the confirmation that this problem still has no definitive solution, and continues to be a relevant challenge for developers, since its solution is closely linked to the quality of the code generated by the compilers in general.

CCS CONCEPTS

•Software and its engineering → Compilers;

KEYWORDS

register allocation, instruction scheduling, code optimization

ACM Reference format:
João F. N. Carvalho, Bruno L. Sousa, Marcus R. Araújo, and Mariza A. S. Bigonha. 2017. The Register Allocation and Instruction Scheduling Challenge. In *Proceedings of SBLP 2017, Fortaleza, CE, Brazil, September 21–22, 2017,* 9 pages.
DOI: 10.1145/3125374.3125380

SBLP 2017, Fortaleza, CE, Brazil

DOI: 10.1145/3125374.3125380

1 INTRODUCTION

The performance of a program may be improved if the interdependence between register allocation and instruction scheduling is properly administrated. The register allocation and the instruction scheduling problems have been extensively studied, both individually and jointly [1, 3, 4, 6, 8–10, 13, 14, 16–20, 26, 29–35, 37, 39, 41].

The purpose of register allocation is to map each temporary variable of the intermediate code to a physical register of the target machine. Register allocation may be seen as accomplished in two phases: *allocation*, which decides which values should be in registers, and *assignment*, which assigns a specific register to each value. In this paper, the term register allocation will indicates both phases. The most widespread method to solve the register allocation problem is the graph coloring [5]. Like the graph coloring, the register allocation is NP-Complete [10, 11].

Instruction scheduling aims at increasing the rate of instructions executed per machine cycle. To achieve it, the scheduler reorders intermediate code instructions as long as it satisfies data dependencies, control dependencies, and structural dependencies. Loop Scheduling orders instructions inside a loop body. Local Scheduling orders instructions inside basic blocks. Modulo Scheduling [38] and List Scheduling [12, 19] are the most used techniques for loop scheduling and local scheduling, respectively. The problem of optimal instruction scheduling is NP-Hard [26, 27].

Register allocation and instruction scheduling tasks are interdependent, that is, they interfere with each other, being involved in a prioritization problem: if the allocation is done before scheduling, *postpass method*, false dependencies between statements may be introduced, limiting the opportunities for reordering instructions and therefore the possibility to execute them in parallel. On the other hand, if scheduling is performed prior to allocation, *prepass method*, the pressure on the registers may rise, increasing the amount of values spills into memory. Examples are given in [2, 37].

This paper presents a *Systematic Literature Review* (SLR) on the interdependence between *register allocation* (RA) and *instruction scheduling* (IS). This SLR has as main objectives: (i) to evaluate the effect of the interdependence between register allocation and instruction scheduling regarding code optimization; (ii) to present how the literature has dealt with the execution of these two tasks together; (iii) to provide a large-scale study to better understand

this problem. To achieve these gols, three research questions were proposed and will be answered in Section 2.3.2.

2 SISTEMATIC LITERATURE REVIEW

For Kitchenham and Charters [24], a *Systematic Literature Review* is "a means of identifying, assessing and interpreting all available evidence relevant to a specific issue, thematic area, or phenomenon of interest". The studies identified by a SLR may be classified as *primary studies* while the SLR itself is a form of *secondary study*, since it reviews all the primary studies to gather and synthesize evidence of a particular research subject. For the authors, the objectives of an SLR are: (i) to summarize the technology or area studied, in order to know its limitations and benefits; (ii) discover gaps in the studies of some technology or research areas that have not yet been completed and suggest them for future research; (iii) provide new structures, directing new areas of research; (iv) studying technologies and theories in order to confirm theses or raise new hypotheses for studies.

The SLR presented in this paper addresses studies about the interdependence between register allocation and instruction scheduling problem. It was performed in three phases: *planning*, *execution* and *analysis*. Section 2.1 presents the planning phase. Section 2.2 describes the execution, presenting the steps and the results of the selection process of the primary studies. Section 2.3 performs the analysis, it shows the most relevant points of some selected papers and answers the research questions.

2.1 Planning Phase

In the planning phase, all the necessary elements for the elaboration of the SLR were defined: the research questions to be investigated, the search string to be used, the databases to be searched and the inclusion and exclusion criteria of the primary studies.

2.1.1 Research Questions (RQ). We propose the following research questions to support this study:

- RQ1: What are the main approaches available in the literature to deal with the interdependence between register allocation and instruction scheduling?
- RQ2: Does a joint approach of register allocation and instruction scheduling tasks really improve code optimization?
- RQ3: For what kinds of architectures has the register allocation and instruction scheduling been performed together?

2.1.2 Search String. The search string is used to search the selected databases for the primary studies. The first step to define the search string is to identify the most relevant keywords related to the research questions, as well as the synonyms of these keywords. In this SLR, the keywords and synonyms chosen were as follows:

- interdependence: *interdependence*, *interconnection*, *interrelationship*, *linkage*, *association*, *connection*, *correlation*, *relationship* , *combining*.
- instruction scheduling: *instruction scheduling*, *scheduling*.
- register allocation: *register allocation*, *variable spilling*, *register packing*, *register pressure*, *register tiling*.
- register allocation and instruction scheduling: *register allocation and instruction scheduling*.

Next, the logical operators `AND` and `OR` were used to connect these words. The `OR` operator was used to connect all the synonyms of each keyword, forming groups of connected words. The `AND` operator was used to connect these groups of words. In this way, the following search string had been defined:

> (("interdependence" OR "interconnection" OR "interrelationship" OR "linkage" OR "association" OR "connection" OR "correlation" OR "relationship" OR "combining") AND ("instruction scheduling" OR "scheduling") AND ("register allocation" OR "variable spilling" OR "register packing" OR "register pressure" OR "register tiling")) OR ("register allocation and instruction scheduling")

2.1.3 Databases. The databases chosen for the collection of the primary studies are listed in Table 1. They were chosen because they are electronic databases that have a large collection of full papers and technical reports published at conferences and journals relevants to the academic community.

Database	Address
ACM Digital Library	http://dl.acm.org/
IEEE	http://ieeexplore.ieee.org/
Science Direct	http://www.sciencedirect.com/
Scopus	http://scopus.com/
Springer	http://link.springer.com/
Web of Science	http://webofknowledge.com/

Table 1: Selected electronic databases.

2.1.4 Inclusion and Exclusion Criteria. The inclusion and exclusion criteria allow to classify each paper found in the surveys as a candidate to be included or excluded from the SLR, so that the papers may be restricted to the topic explored. Table 2 presents the defined inclusion and exclusion criteria used in this work.

Inclusion Criteria	Exclusion Criteria
Papers published in Computer Science.	Duplicate papers.
Papers written in English.	Studies classified as tutorials, posters, panels, lectures, round tables, workshops, theses and dissertations.
Papers available in electronic format.	Papers that could not be located.
Papers published in conferences or magazines.	
Papers related to the terms of the search string.	

Table 2: Inclusion and exclusion criteria.

2.2 Execution Phase

The execution phase, obeying the planning described in Section 2.1, consists in the application of the search string in order to search the chosen databases for the primary studies. Besides that, the inclusion and exclusion criteria of the primary studies are applied.

2.2.1 Search Process. In the selected electronic bases, the search process is automated by an existing search engine in the base itself. In each of them, the search was performed using the search string exhibited in Section 2.1.2 and the results obtained were exported. The entire search process was conducted from October 19-21, 2016.

Table 3 shows, for each electronic base searched, the number of the primary studies found. The search process obtained 542 studies.

Database	Total of Studies
ACM Digital Library	134
IEEE Xplore	0
Science Direct	313
Scopus	27
Springer	34
Web of Science	34
Total of Studies Obtained	**542**

Table 3: Primary studies obtained.

The *IEEE Xplore* database did not return any document in the search performed with the defined search string. Unsatisfied with this result, new searches were made keeping the keywords defined, but changing the positions of the logical connectors `AND` and `OR` of the search string. In some cases, the search with the modified string produced results. However, they were all contained in the results of the searches made on the other bases, except for a single article. Due to this non-conformity of the process and, mainly, due to the low contribution that this modification brought to the SLR results, the original search result at the *IEEE Xplore* base was accepted.

2.2.2 Selection Process. After the completion of the search process, the selection of the primary studies started. To perform the evaluation of the studies, the inclusion and exclusion criteria defined in the planning phase were used, and the evaluation was done according to the four steps described as follows. In each step, exclusion criteria were applied, with the objective of eliminating studies that were not useful for the SLR. All studies related to the proposed research theme were maintained.

Step 1. All study were compared to each other. Studies with same title and authors were discarded. This step eliminated 53 repeated studies.

Step 2. The titles of all studies were checked. Studies with titles not related to the research theme were disregarded. In cases where the simple reading of the title was not enlightening, the abstract was read. This step eliminated 434 studies.

Step 3. All abstracts from the remaining studies were read. Articles whose abstracts indicated lack of relevance to the research topic were disregarded. In some cases, the lack of pertinence was only partial, that is, the study dealt with only register allocation, or only with instruction scheduling, without considering the two tasks together and the interdependence between them. In these cases, the paper was not considered. This step discarded 22 studies.

Step 4. Each study was checked considering the exclusion criteria, see Section 2.1.4. Studies that fit into one of these criteria were disregarded. In addition, was verified the availability of each stydy, in order to read it. At this step, eight studies were disregarded, five were PhD theses and the other three were not located, despite all efforts in this direction.

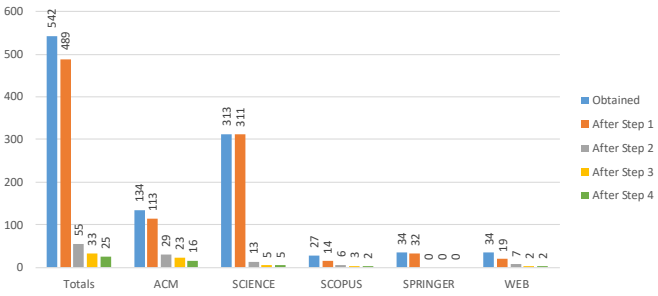

Figure 1: Results of the steps in the analysis process.

Figure 1 presents the results of the selection process, showing the total number of studies at the end of each step for the bases considered in this work, except for the IEEE base. IEEE base does not appear in this figure, because it did not return results in the search process. At the end of the process, remained 25 studies, whose subjects were considered pertinent to the research topic. It is interesting to note that none of the selection step considered the year of publication of the studies, so that no study was excluded or included depending on its date of publication.

2.2.3 Final Result. This section presents the final result of the execution phase. Among the 542 primary studies initially obtained, 25 studies were closely related to the research theme of this SLR, that is, they dealt with the tasks of register allocation and instruction scheduling, as well as the interference that they exert between each other. These 25 studies were selected to analysis phase.

Figures 2 and 3 display the totals of primary studies selected by base, and by decade of publication, respectively. It is interesting to note that between the years 2010 and 2016, a total of nine papers were published, an approximate average of one paper per year.

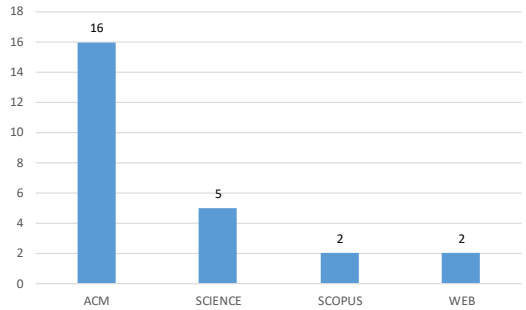

Figure 2: Primary studies by database.

Table 4 lists the 25 papers selected in the execution phase. For each of them, the table shows the number and the title, the basis on which it was found, the publication year and the number of citations in google scholar. These papers were listed in ascending order of their publication year.

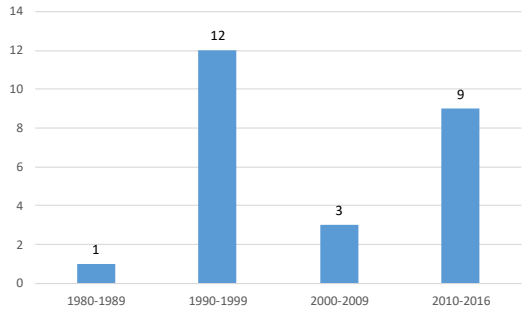

Figure 3: Primary studies per decade of publication.

2.3 Analysis Phase

In this phase, with the data extrated from the 25 papers selected in the execution phase, shown in Table 4, we answer and analyze the results of the proposed research questions defined in Section 2.1.1.

2.3.1 Data Extraction. To perform the data extraction, all the selected papers were read and summarized. Due to lack of space, not all summaries were included here, but they are available at https://goo.gl/DzeAXn. Here, we highlight briefly only the most relevant points of the papers published from 2010 to October 2016.

Prior the presentation of the summaries, it is important highlight that: (1) two main approaches have been used to solving the problem of the interdependence between RA and IS: the *cooperative* and the *integrated* approaches. In the integrated approach, register allocation and instruction scheduling are done simultaneously. In the cooperative approach, the allocation and scheduling are done separately, but there is an exchange of information between them, so that one can take into account the needs of the other. (2) Most of the selected studies consider local optimizations, that is, within basic blocks, and few of them consider global optimizations.

As an example of an integrated approach, we have the Unison tool presented in [29], which solves allocation and scheduling problems using combinatorial optimization. Pinter [37] shows an example of a cooperative approach, using the *parallelizable interference graph*, which is the only data structure used by register allocator and instruction scheduler. These two approaches are widely cited.

Out-of-order processors were the focus of the work of Govindarajan et al. [14]. They propose an integrated approach, based on a data dependency graph, which uses heuristics to perform the scheduling having the number of registers as reference. The authors did not report any test realized with this approach.

Kri and Feeley [26], use *genetic algorithms* to propose an integrated approach of register allocation and instruction scheduling. The fitness function chosen was the code execution time. The model aims at minimizing this function. Due to this choice, the modeling required little information about the target architecture, so that the real effects of the optimization could be considered, including indirect effects, making the results more homogeneous.

Cutcutache and Wong [8] proposed an integrated approach to reduce the compilation time of the programs. An algorithm is used to divide the lifetime of the variables according to the regions of the program in which it is located. The tests measured the compilation time and the performance of the scheduling and allocation tasks on the OpenIMPACT compiler. The results indicated a reduction of 50% in the execution time of these tasks.

Kim and Lee [19] proposed a new allocation technique combining the advantages of the coloring graphs [5] and fine-grain[1] [25]. In addition, they present a new technique, Integrated Register Allocation and Instruction Scheduling (IRIS), to integrate the allocation

Number. Title	Base	Year	Citations
1. Code Scheduling and Register Allocation in Large Basic Blocks	ACM	1988	295
2. Integrating Register Allocation and Instruction Scheduling for RISCs	ACM	1991	208
3. A Novel Framework of Register Allocation for Software Pipelining	ACM	1993	121
4. Resource Spackling - A Framework for Integrating Register Allocation in Local and Global Schedulers	WEB	1994	48
5. Scheduling with Register Constraints for DSP Architectures	SCIENCE	1994	6
6. Combining Register Allocation and Instruction Scheduling	ACM	1995	85
7. CRAIG: A Practical Framework for Combining Instruction Scheduling and Register Assignment	SCOPUS	1995	36
8. Register Allocation Sensitive Region Scheduling	SCOPUS	1995	16
9. Register Allocation with Instruction Scheduling: A New Approach	WEB	1996	166
10. Using Integer Linear Programming for Instruction Scheduling and Register Allocation in Multi-issue Processors	SCIENCE	1997	43
11. Experiences with Cooperating Register Allocation and Instruction Scheduling	ACM	1998	9
12. Scheduling Expression DAGs for Minimal Register Need	SCIENCE	1998	36
13. Evaluating Register Allocation and Instruction Scheduling Techniques in Out-Of-Order Issue Processors	ACM	1999	12
14. Minimum Register Instruction Scheduling: A New Approach for Dynamic Instruction Issue Processors	ACM	2000	5
15. Genetic Instruction Scheduling and Register Allocation	ACM	2004	12
16. Fast, Frequency-based, Integrated Register Allocation and Instruction Scheduling	ACM	2008	9
17. Fine-Grain Register Allocation and Instruction Scheduling in a Reference Flow	ACM	2010	3
18. On Minimizing Register Usage of Linearly Scheduled Algorithms with Uniform Dependencies	ACM	2010	2
19. Register Allocation with Instruction Scheduling for VLIW-Architectures	ACM	2010	4
20. Minimizing Schedule Length via Cooperative Register Allocation and Loop Scheduling for Embedded Systems	ACM	2011	1
21. Variable Assignment and Instruction Scheduling for Processor with Multi-module Memory	SCIENCE	2011	4
22. Constraint-Based Register Allocation and Instruction Scheduling	ACM	2012	25
23. Register Allocation for Fine Grain Threads on Multicore Processor	SCIENCE	2015	0
24. Register Allocation and Instruction Scheduling in Unison	ACM	2016	1
25. Register Allocation and Promotion Through Combined Instruction Scheduling and Loop Unrolling	ACM	2016	7

Table 4: Final result of the execution phase.

[1] Technique in which the allocation for a value is made each time the value is referenced.

and the scheduling tasks. The tests, comparing IRIS with the conventional postpass technique, showed that the compilation time was improved on average 38%.

Philippidis and Shang [35] presented an approach to minimize the register pressure in nested loops using *linear scheduling* and *loop unrolling*. The linear scheduling is used to move instructions inside the loop, so it functions as a specific form of instruction scheduling. The results show that the technique has a great potential and may be useful for applications of different areas.

Ivanov [17] tries to combine allocation and scheduling taking into account the characteristics of static scheduling for VLIW architectures. The term *web* is used to denote a set of nodes and edges of the control flow graph of a procedure, whose execution requires that intermediate values are stored. Allocation and scheduling are carried out using the identified webs. The tests carried out indicate a good performance of this approach in relation to the allocation strategy in scheduled codes.

To minimize the time spent of loop scheduling, Huang et al. [16] propose the Cooperative Re-scheduling Register Allocation (CRRA), which uses *rotation scheduling* [7] as a technique to manipulate the loops. The register allocator itself balances the stages of scheduling and allocation. The results show that this technique may, on average, reduce by 12% the time spent by scheduling in relation to other approaches, such as Register Constrained Rotation Scheduling [40].

Zhang et al. [41] propose Performance and Energy Optimal Scheduling (PEOS) framework, an integer linear programming model, to optimize the performance and energy consumption of multi-module memory. Addressing the IS and RA simultaneously, it aims at generating a schedule with a minimum length and minimum energy consumption. The tests performed showed that the technique is promising, it presented better results in all comparisons carried out and has showed to be able to achieve a good compromise between performance and consumption power.

Lozano et al. [30] treat the global allocation and the local scheduling as *constraint programming*. The problem is modeled considering the program instructions, the characteristics of the target architecture and the specific constraints for allocation and scheduling. A new form of representing the intermediate code, Linear Static Single Assigment (LSSA), was used. The results indicate that the quality of the generated code is similar to that of the LLVM code.

Kiran et al. [20] proposed two heuristics for register allocation in multicore architectures. From the identification and extraction of the regions with parallelism, or *fine grained threads* [22], the dependency graph between these regions is constructed and the scheduling for each processor core is done [21, 23]. The allocation is done by applying the coloring technique [5] in the created dependency graph. The proposed heuristics were more efficient than the methods developed for unicore processors.

Lozano et al. [29] propose a tool, Unison, based on combinatorial models, as an integrated approach of solution. The tests performed indicated that, for simple architectures, the generated code has similar quality to the code generated by LLVM. For complex architectures, the quality is higher than that generated by LLVM.

Domagala et al. [10] focus their research on the optimization of RA for the loops of the program. They presented and implemented in the Open64 compiler an approach to register allocation and register promotion [28] that considers the impacts of IS and loop unrolling. The tests indicated a substantial improvement in the number of spills for many loops with high register pressure.

2.3.2 Answers to Research Questions. After organizing the data extracted from the selected papers, and gather all the information needed, the research questions were answered.

RQ1. What are the main approaches available in the literature to deal with the interdependence between register allocation and instruction scheduling?

The primary studies analyzed pointed out that the main approaches to dealing with the interdependence between RA and IS are the cooperative and the integrated approaches.

Figures 4 and 5 present two graphs related to the classification of the approaches used in the papers that make up the result of this SLR. The most used approach is the cooperative, appearing in 56% of the articles. The integrated approach was used in 40% of the papers. Valluri and Govindarajan [39] only evaluated existing solutions and does not seek to solve the problem with a specific approach. For this reason, it appears as *Not Applicable* in the graphics.

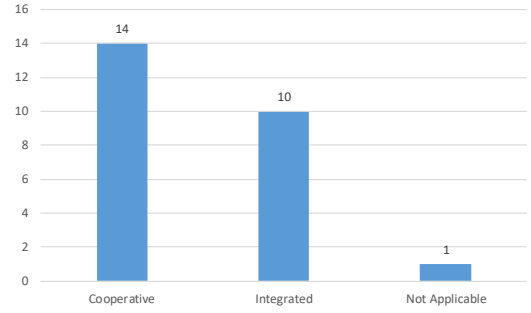

Figure 4: Studies by type of approach.

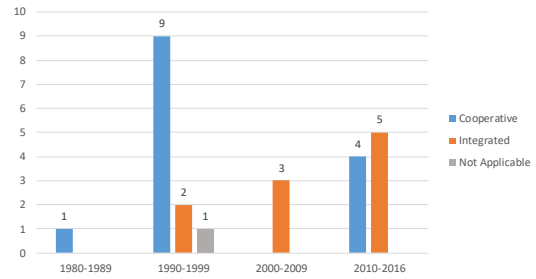

Figure 5: Types of approach per decade of publication.

In Bradlee et al. [3], the IPS and RASE methods, which handle the register allocation and instruction scheduling in *joint approach*[2], have proved to be better to solve the problem of interference between these two tasks than the postpass method[3], which treats these taks separately. However, the RASE method, which addresses the problem in an integrated way, did not bring gains that would justify its use instead of the IPS method, which uses a cooperative

[2] An *integrated* or *cooperative* approach.
[3] Described in [12, 15].

approach. The authors, themselves, indicate the IPS method. Thus, we compute this paper as a *cooperative* approach.

Summary: the primary studies analyzed suggest that the main approaches to dealing with the interdependence between register allocation and instruction scheduling are: *cooperative* and *integrated*. The first one is the most used.

RQ2. Does a joint approach of register allocation and instruction scheduling tasks really improve code optimization?

In the SLR performed, a joint approach improved the code optimization in 80% of the papers studied. Figure 6 displays the graph with the totals for each of the answer options to this research search. We consider that only three articles did not present better optimized code using some type of joint approach. This research question is not applicable for two articles.

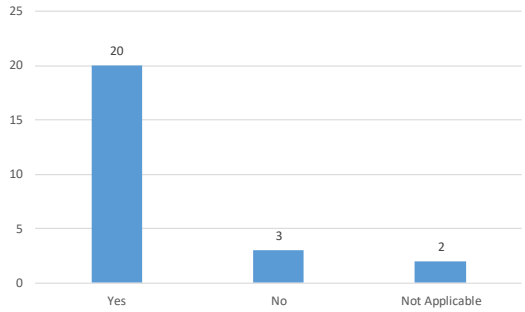

Figure 6: Answers to RQ2.

The works of Chang et al. [6] and Zhang et al. [41] are two cases where a integrated approach was used to solve the problem without improvement in code optimization. They describe integrated approaches solutions using integer linear programming. In both cases, the time to obtain the solutions was very high, making the practical use of the proposed techniques unfeasible.

Valluri and Govindarajan [39] tested four solution approaches[4] in an out-of-order processor and report that the cooperative approaches tested did not improve code optimization. In the tests, a simulator was used instead of a real machine. According to the authors, for out-of-order processors, the code generation process must avoid pressure on the registers and spills into memory, without worrying about the instruction scheduling task. In addition, they found that the cooperation between the register allocation and the instruction scheduling showed insignificant improvements in performance, when compared with prepass and postpass techniques, without any kind of integration. For benchmarking with high pressure on the registers, the evaluated postpass and postpass-like[5] techniques generated better performance codes, as they prioritized the register allocation. For the authors, these results are due to the characteristics of the out-of-order processor that are able to dynamically schedule instructions.

This RQ2 is not applicable to papers of Govindarajan et al. [14] and Depuydt et al. [9]. In the first, the authors proposed an integrated approach for out-of-order processors. However, such approach was not evaluated and was listed as a future work in their paper. In the second article, comparative results between the method created and other methods were not presented.

Among the types of joint approaches, the cooperative approach was the one that produced the greatest number of cases in which improvement of code optimization was observed. In 93% of cases using this approach, more efficient code was produced, whereas in cases where the integrated approach was used, code improvement occurred in 78% of cases. Figure 7 presents the totalizations of the answers to this research question by the type of approach used.

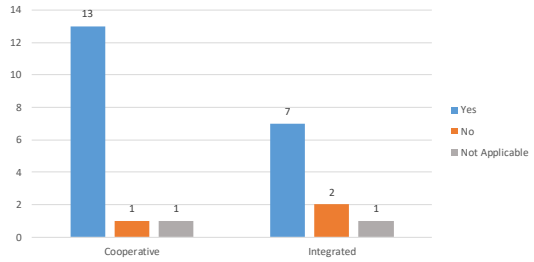

Figure 7: Answers to RQ2 by type of approach.

Finally, it is important to highlight that some of the joint approaches considered in the statistics of this research question have certain restrictions. In the work of Norris and Pollock [33], the cooperative strategy RASER produces good results only for architectures with few registers. For machines with a high number of processors, many scheduling opportunities are lost and the generated code does not perform well. The algorithm created by Kessler [18] handles instruction scheduling locally and works only with small and medium basic blocks, with a maximum of 50 instructions. For larger basic blocks, the algorithm should not be used.

Summary: the answer to RQ2 is affirmative, since in most of the studies returned in this SLR a joint approach improves code optimization.

RQ3. For what kinds of architectures has the register allocation and instruction scheduling been performed together?

Every object program is generated for a particular architecture. Thus, when talking about the code generation, one must consider the characteristics of the architecture for which the code is being generated. In this sense, considering the machine architectures, studies show that the problem of mutual interference between register allocation and instruction scheduling tasks is especially important for the Reduced Instruction Set Computer (RISC), superscalar and all the new architectures.

In general, post-RISC architectures use simple, fixed-length instructions. They have a large number of registers and allow the execution of several instructions per cycle using the pipelining technique[6]. They have several functional units which allows the exploration of instruction level parallelism. Memory access is made exclusively by specific instructions, such as *load* and *store*. The computation instructions may operate only on the registers, being unable to access the memory directly. Since all these features are

[4]Presented in [12, 13, 36].
[5]Method described in [36].
[6]Allows the simultaneous execution of different parts or stages of instructions.

available to compilers of these architectures, these compilers need to use them correctly to generate efficient execution codes.

Most of the papers studied in this SLR considered architectures following the general model of RISC, however, some papers use specific architectures, or indicate some particular characteristics.

Berson et al. [1], Brasier et al. [4], Ivanov [17] use Very Long Instruction Word (VLIW) or Longtext Instruction Word (LIW) architectures. The difference between the terms VLIW and LIW is quite arbitrary and generally they describe the same type of architecture. It is an architecture that encodes multiple operations into a single instruction, and each operation runs in parallel on one of the processor's execution units. The guarantee of independence between these instructions should be given by the compiler.

In the works of Govindarajan et al. [14], Valluri and Govindarajan [39], the proposed approaches are for out-of-order processors. Kiran et al. [20] consider the allocation and scheduling problem facing multicore processors with *fine grain parallelism*[7] and perform tests using dual core and quad core processors. In Kim and Lee [19], the authors do not mention whether the proposed approach is focused on a specific processor type. However, during the evaluation part, the implementation has used the ARM7TDMI processors. In Pinter [37], the method is developed using a generic model of a RISC architecture. The tests were executed on a Pentium processor with two nonsymmetric pipelines that share functional units. The method presented by Depuydt et al. [9] aimed at ASIP architecture processors, widely used in embedded systems for real-time Digital Signal Processing (DSP) applications. In Zhang et al. [41], the method developed is intended to optimize the performance and power consumption of multi-module memory used in processors oriented to DSP applications.

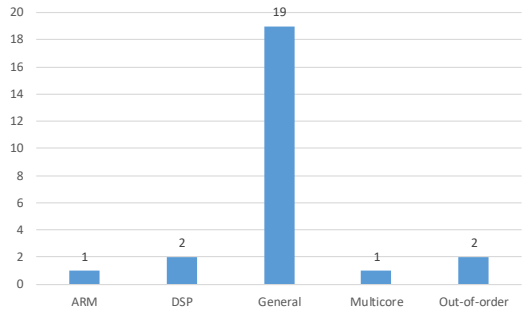

Figure 8: Total studies by type of architecture.

Summary: the graph of Figure 8 answers the RQ3. It shows the total of papers by type of architecture. The architectures, cited by the articles, that resemble RISC architecture, such as the VLIW, LIW, and superscalar architectures, but that were not associated with a specific processor type, are grouped under the term *General*. From this graph, it may be seen that 76% of the articles considered are related to *General* architecture, while 24% of them are related to other types of architecture.

[7]Division of a program into a large number of tasks, each one with few instructions.

3 THREATS TO VALIDITY

This section discusses the main threats to the validity of this SLR and presents the decisions taken to mitigate them.

Definition of the Search String. The search string must be defined very well in order to return studies relevant to the search topic. In this SLR, several synonyms refering to the main terms of the review objective were searched. Some pilot searches were conducted in order to find new synonyms for the search string. Thus, it is expected that the defined search string has returned as many relevant papers as possible. But it is not possible to state that all the studies concerning relations between the interdependence between RA and IS has been returned.

Databases. The choice of electronic databases is another factor that may impact the results of a SLR. Six different electronic databases were selected for the search process, but studies relevant to this review may exist in other databases not used in this survey. To mitigate this threat, it was selected databases with a large collection of full papers published as articles or technical reports at conferences or journals relevants to the academic community.

Selection Process. Two steps in the selection process were based on metadata analysis, that is, on the analysis of the titles and abstracts of the studies. Thus, relevant studies whose metadata did not show a clear relation with the SLR theme may have been disregarded. To minimize this threat, the evaluation for inclusion or exclusion of the studies was divided among the authors of this paper. Each one of them evaluated a set of primary studies, and the results obtained were validated by the other authors. It is important to highlight that the exclusion of theses and dissertations does not must be a problem, because, normally, one or more papers are produced from these works. Finally, this SLR considered only papers written in English. It is possible that some relevant studies may be written in another idiom.

4 CONCLUSION

This paper presented a *Systematic Literature Review* (SLR) on the problem of the interdependence between register allocation and instruction scheduling. This is a complex and very important problem related to the generation of good quality compiler code.

Traditionally, two main approaches deal with this problem: the cooperative and the integrated approaches. In this SLR, the first one appears in 56% of the studies, while the last one is used in 40%. The use of some joint approach, whether cooperative or integrated, improved the code optimization in 80% of the papers studied. Looking at these approaches separately, it was observed that the cooperative one improved the code optimization in 93% of cases, while the integrated one in 78% of cases. Considering the architectures, around 76% of studies are related to RISC-like architectures, and 24% are related to other kinds of architectures.

Observing the graph of Figure 5, it is clear that over the last decades researchers have focused on the integrated approach. Between 2000 and 2016, eight of 12 studies analyzed used integrated approach. No cooperative approaches was investigated between 2000 and 2009, but they became again an object of interest and appear in almost half of the researches performed between 2010 and 2016. Thus, while, on the one hand, the integrated approach was more explored over the two last decades, on the other hand, it

also gave space to the cooperative one in recent years. Besides that, as shown in the graph of Figure 7, the integrated approach had a little less success than the cooperative ones.

To avoid misleading interpretations, it is important to highlight that each primary study was analyzed individually in this work, meaning that, the approaches of the different studies were not compared with each other regarding time or any other aspect. The categorization of these approaches was made in a very general way, as cooperative or integrated, and the statistics associated were presented. Concluding, this SLR showed tendencies among the choices made to deal with the interdependence between RA and IS problem but did not suggest that an approach is better then other.

The main contributions of this work are: (1) it provided a large-scale study to better understand the interdependence problem between register allocation and instruction scheduling and the main approaches proposed to solve it until now; (2) our results showed that, although it is an old problem, there is no consensus on it solution neither on how to address it; (3) it showed that this research field is still open to new solutions and has been the subject of recent researches [10, 16, 17, 19, 20, 29, 30, 35, 41].

As future work, some improvements may be done: (1) redefinition of the search string by identifying and excluding ineffective keywords and including new ones; (2) inclusion of other databases, for instance google scholar for primary studies; (3) verification in the selected papers, through a list of references, their citations in order to find other relevant studies that had not been returned; (4) inclusion of new analysis like optimality, type of problem, modeling, execution time, number of software tools implementing the paper ideas, and so one, in order to facilitate comparations among different approaches; (5) identification of the circumstances that has made one approach be better than another; (6) quantification of the impact of each study over the subsequent works in order to identify the most relevant studies and then refine the selection process considering this impact.

REFERENCES

[1] D. A. Berson, R. Gupta, and M. L. Soffa. 1994. Resource Spackling - A Framework for Integrating Register Allocation in Local and Global Schedulers. In *Parallel Architectures and Compilation Techniques (IFIP Transactions A-Computer Science and Technology)*, M Cosnard, GR Gao, and GM Silberman (Eds.), Vol. 50. 135–145.

[2] Mariza A. S. Bigonha. 1992. Geração e otimização de código: Levantamento dos problemas e restrições impostas pelas arquiteturas RISC e indicativos de soluções. *Série de Monografias em Ciência da Computação 10/92* (1992).

[3] David G. Bradlee, Susan J. Eggers, and Robert R. Henry. 1991. Integrating Register Allocation and Instruction Scheduling for RISCs. *SIGARCH Comput. Archit. News* 19, 2 (1991), 122–131.

[4] T. S. Brasier, P. H. Sweany, and S. J. Beaty. 1995. CRAIG: A Practical Framework for Combining Instruction Scheduling and Register Assignment. *Parallel Architectures and Compilation Techniques - Conference Proceedings* (1995), 11–18.

[5] Gregory J Chaitin, Marc A Auslander, Ashok K Chandra, John Cocke, Martin E Hopkins, and Peter W Markstein. 1981. Register allocation via coloring. *Computer languages* 6, 1 (1981), 47–57.

[6] Chia-Ming Chang, Chien-Ming Chen, and Chung-Ta King. 1997. Using Integer Linear Programming for Instruction Scheduling and Register Allocation in Multi-issue Processors. *Computers & Mathematics with Applications* 34, 9 (1997), 1–14.

[7] Liang-Fang Chao and Andrea LaPaugh. 1993. Rotation Scheduling: A Loop Pipelining Algorithm. In *Proceedings of the 30th International Design Automation Conference (DAC '93)*. 566–572.

[8] Ioana Cutcutache and Weng-Fai Wong. 2008. Fast, Frequency-based, Integrated Register Allocation and Instruction Scheduling. *Softw. Pract. Exper.* 38, 11 (2008), 1105–1126.

[9] Francis Depuydt, Gert Goossens, and Hugo De Man. 1994. Scheduling with Register Constraints for DSP Architectures. *Integration, the VLSI Journal* 18, 1 (1994), 95–120.

[10] Lukasz Domagala, Duco van Amstel, Fabrice Rastello, and P. Sadayappan. 2016. Register Allocation and Promotion Through Combined Instruction Scheduling and Loop Unrolling. In *Proceedings of the 25th International Conference on Compiler Construction (CC 2016)*. 143–151.

[11] M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[12] Philip B. Gibbons and Steven S. Muchnick. 1986. Efficient Instruction Scheduling for a Pipelined Architecture. *SIGPLAN Not.* 21, 7 (1986), 11–16.

[13] James. R. Goodman and Wei-Chung Hsu. 1988. Code Scheduling and Register Allocation in Large Basic Blocks. In *Proceedings of the 2Nd International Conference on Supercomputing (ICS '88)*. 442–452.

[14] Ramaswamy Govindarajan, Chihong Zhang, and Guang R. Gao. 2000. Minimum Register Instruction Scheduling: A New Approach for Dynamic Instruction Issue Processors. In *Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing (LCPC '99)*. 70–84.

[15] John L. Hennessy and Thomas Gross. 1983. Postpass Code Optimization of Pipeline Constraints. *ACM Trans. Program. Lang. Syst.* 5, 3 (1983), 422–448.

[16] Yazhi Huang, Qingan Li, and Chun Jason Xue. 2011. Minimizing Schedule Length via Cooperative Register Allocation and Loop Scheduling for Embedded Systems. In *Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM '11)*. 1038–1044.

[17] D. S. Ivanov. 2010. Register Allocation with Instruction Scheduling for VLIW-Architectures. *Program. Comput. Softw.* 36, 6 (2010), 363–367.

[18] Christoph W. Kessler. 1998. Scheduling Expression DAGs for Minimal Register Need. *Computer Languages* 24, 1 (1998), 33–53.

[19] Dae-Hwan Kim and Hyuk-Jae Lee. 2010. Fine-Grain Register Allocation and Instruction Scheduling in a Reference Flow. *Comput. J.* 53, 6 (2010), 717–740.

[20] D.C. Kiran, S. Gurunarayanan, Janardan P. Misra, and Munish Bhatia. 2015. Register Allocation for Fine Grain Threads on Multicore Processor. *Journal of King Saud University - Computer and Information Sciences* (2015), 1–8.

[21] D. C. Kiran, S. Gurunarayanan, Faizan Khaliq, and Abhijeet Nawal. 2012. *Compiler Efficient and Power Aware Instruction Level Parallelism for Multicore Architecture*. Springer Berlin Heidelberg, Berlin, Heidelberg, 9–17.

[22] D. C. Kiran, S. Gurunarayanan, and J. P. Misra. 2011. Taming Compiler to Work with Multicore Processors. In *2011 International Conference on Process Automation, Control and Computing*. 1–6.

[23] D. C. Kiran, B. Radheshyam, S. Gurunarayanan, and J. P. Misra. 2011. Compiler Assisted Dynamic Scheduling for Multicore Processors. In *2011 International Conference on Process Automation, Control and Computing*. 1–6.

[24] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-001. Keele University and Durham University Joint Report.

[25] Priyadarshan Kolte and Mary Jean Harrold. 1993. Load/Store Range Analysis for Global Register Allocation. *SIGPLAN Not.* 28, 6 (June 1993), 268–277.

[26] Fernanda Kri and Marc Feeley. 2004. Genetic Instruction Scheduling and Register Allocation. In *Proceedings of the The Quantitative Evaluation of Systems, First International Conference (QEST '04)*. 76–83.

[27] R. Leupers. 2000. *Code Optimization Techniques for Embedded Processors: Methods, Algorithms, and Tools*. Kluwer Academic Publishers, Norwell, MA, USA.

[28] Raymond Lo, Fred Chow, Robert Kennedy, Shin-Ming Liu, and Peng Tu. 1998. Register Promotion by Sparse Partial Redundancy Elimination of Loads and Stores. *SIGPLAN Not.* 33, 5 (May 1998), 26–37.

[29] R. C. Lozano, M. Carlsson, G. H. Blindell, and C. Schulte. 2016. Register Allocation and Instruction Scheduling in Unison. In *Proceedings of the 25th International Conference on Compiler Construction (CC 2016)*. 263–264.

[30] R. C. Lozano, M. Carlsson, F. Drejhammar, and C. Schulte. 2012. Constraint-Based Register Allocation and Instruction Scheduling. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming - Volume 7514*. 750–766.

[31] Rajeev Motwani, Krishna V. Palem, Vivek Sarkar, and Salem Reyen. 1995. Combining Register Allocation and Instruction Scheduling. *Courant Institute, New York University* (1995).

[32] Qi Ning and Guang R. Gao. 1993. A Novel Framework of Register Allocation for Software Pipelining. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '93)*. 29–42.

[33] Cindy Norris and Lori L. Pollock. 1995. Register Allocation Sensitive Region Scheduling. *Parallel Architectures and Compilation Techniques - Conference Proceedings* (1995), 1–10.

[34] Cindy Norris and Lori L Pollock. 1998. Experiences with cooperating register allocation and instruction scheduling. *International journal of parallel programming* 26, 3 (1998), 241–283.

[35] Cesar J. Philippidis and Weijia Shang. 2010. On Minimizing Register Usage of Linearly Scheduled Algorithms with Uniform Dependencies. *Comput. Lang. Syst. Struct.* 36, 3 (2010), 250–267.

[36] Shlomit S. Pinter. 1993. Register Allocation with Instruction Scheduling. *SIGPLAN Not.* 28, 6 (June 1993), 248–257.

[37] Shlomit S. Pinter. 1996. Register Allocation with Instruction Scheduling: A New Approach. *Journal of Programming Languages* 4, 1 (1996), 21–38.

[38] B. Ramakrishna Rau. 1994. Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops. In *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO 27)*. ACM, New York, NY, USA, 63–74.

[39] Madhavi Gopal Valluri and R. Govindarajan. 1999. Evaluating Register Allocation and Instruction Scheduling Techniques in Out-Of-Order Issue Processors. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT '99)*. 78–83.

[40] Kaisheng Wang, Ted Zhihong, Yu Edwin, and H. M. Sha. 1998. RCRS: a framework for loop scheduling with limited number of registers. In *Proceedings of the 8th Great Lakes Symposium on VLSI (Cat. No.98TB100222)*. 386–391.

[41] Lei Zhang, Meikang Qiu, Edwin H.-M. Sha, and Qingfeng Zhuge. 2011. Variable Assignment and Instruction Scheduling for Processor with Multi-module Memory. *Microprocessors and Microsystems* 35, 3 (2011), 308—317.