Engineering Heritage Journal (GWK)

DOI : http://doi.org/10.26480/gwk.02.2018.04.08

ZIBELINE INTERNATIONAL

Print ISSN : 2521-0904
Online ISSN : 2521-0440

CODEN: EHJNA9

COMPARISON OF INSTRUCTION SCHEDULING AND REGISTER ALLOCATION FOR MIPS AND HPL-PD ARCHITECTURE FOR EXPLOITATION OF INSTRUCTION LEVEL PARALLELISM

Rajendra Kumar*

Vidya College of Engineering, Meerut (India)
****Corresponding Author Email: rajendra04@gmail.com***

ARTICLE DETAILS

Article History:

Received 12 November 2017
Accepted 12 December 2017
Available online 1 January 2018

ABSTRACT

The integrated approaches for instruction scheduling and register allocation have been promising area of research for code generation and compiler optimization. In this paper we have proposed an integrated algorithm for instruction scheduling and register allocation and implemented it for compiler optimization in machine description in trimaran infrastructure for exploitation of Instruction level parallelism. Our implementation in trimaran infrastructure shows that our scheduler reduces the number of active live ranges dealt with linear scan allocator. As a result only few spills were needed and the quality of the code generated was improved. For our experiments we used 20 benchmarks available with trimaran infrastructure for HPL-PD architecture. We compare some of these results with results obtained by Haijing Tang et al (2013) performed by LLVM compiler on MIPS architecture. For our experimental work we added machine description (MDES) targeted to HL-PD architecture. The implemented algorithm is based on subgraph isomorphism. The input program is represented in the form of directed acyclic graph (DAG). The vertices of the DAG represent the instructions, input and output operands of the program, while the edges represent dependencies among the instructions.

KEYWORDS

ILP, Instruction Scheduling, Register Allocation, Trimaran Simulator, Parallelilsm.

1. BACKGROUND OF TOPIC SELECTION

China For exploitation of instruction level parallelism through compiler optimization it is very important to consider the code optimization and code generation phases efficiently. Instruction scheduling and register allocation have great importance in code generation [1]. Recent studies show a lot of efforts on integrated approaches for instruction scheduling and register allocation [2,3]. The combinations of instruction scheduling and register allocation have been discussed in section 3. The key aspect of proposed technique is the modeling of the hardware resources by redefining the MDES of trimaran infrastructure and comparison of results with LLVM compiler [4].

The algorithm is implemented for instruction scheduling and register allocation based on subgraph isomorphism theory on the trimaran compiler for HPL-PD architecture [5,6]. For the purpose of feasibility and flexibility, the integrated approach is designed for the HPL-PD architecture on the trimaran compiler. This paper is organized as follows: Section 2 presents related work, Section 3 presents use of instruction scheduling and register allocation, Section 4 presents HPL-PD Architecture, Section 5 presents proposed algorithm for instruction scheduling and register allocation, Section 6 presents MDES description steps, Section 7 represents the Experimental results, Section 8 represents the comparison of some of the results with MIPS architecture on LLVM compiler, Section 9 represents the conclusion and future scope [7,8].

The advantage of register allocation is the speed. Some researchers pointed that Register allocation is the optimization technique that can increase efficiency of algorithm upto 250% [9]. As the computers have limited number of CPU registers therefore it is not possible to assign all variables to the registers. A 32-bit variable spilled to memory avails an allocation of 32 bit of stack space. Such variable has a much slower processing speed then a variable in register. Another researchers pointed that the spill free register allocation is an NP-Complete problem [10]. They proved that any graph is the interference graph of a program. Graph coloring and Linear scan approaches have been used for register allocation. In a study, the researchers added a recent work to Linear Scan algorithms called Extended Linear Scan [11]. It applies copy and swap instructions along the source program and uses minimal number of registers to compile the program.

The graph coloring is seen as the optimal approach for register allocation [12]. It has been adopted by several modern compilers like LLVM and Trimaran. A simpler approach linear scan has also been used for the same. The fact by which the minimal coloring to a graph was proved to be NP-Complete [13]. Subgraph isomorphism is defined as a generalization of graph isomorphism problem which asks whether a graph *G* isomorphic to graph *H*. A group researcher also computed a subgraph isomorphism problem that has a query complexity of $\Omega(n^{3/2})$ [14]. Subgraph isomorphism is seen as a very general form for pattern matching and it provides a platform for several important graph problem, like Hamiltonian paths, shortest path, cliques, etc [15-17].

2. RELATED WORK

Integer programming based optimal register allocation algorithm have been developed for regular architecture. The recent research on register allocation is based on SSA-form [18,19]. The interference graphs of SSA-form can be colored in polynomial time. SSA (Static Single Assignment) is defined as the property of an intermediate presentation which requires every variable to be assigned only once. Ever variable is to be defined before it is used. The LLVM and trimaran compiler infrastructure use SSA form for all scalar register values in their primary code representation. Some researchers presented steps to convert the intermediate representation in to SSA form [20]. SSA form is optimal and has no unnecessary terms.

Several instruction scheduling has been developed and implemented recently. An instruction scheduling algorithm for the TRIPS architecture is presented presented as an integrated algorithm for instruction selection and register allocation but focused on instruction set with mixed instruction formats for 16- and 32-bit instructions on the ARCompact ISA [21,22]. They got a good code reduction of 16.7% and also a performance gain of 17.7%. This contribution differs from this work because they have an instruction set architecture with two representations of the same instruction and focused on selecting the best one for each case.

An interesting way to see register allocation is as a Multi-Flow of Commodities (MFC) problem. This idea was introduced in order to perform local register allocation [23]. Local allocation is the version of register allocation that is restricted to basic blocks only, in contrast to global register allocation, that is concerned about the whole program. A study discussed that Spill Free Register Allocation has polynomial time solution for SSA-form programs, but it is NP-complete for programs in general [24]. An important breakthrough in register allocation happened in 2005, when three different research groups, proved independently that the interference graphs of programs in Static Single Assignment (SSA) form are chordal [25-27]. A researcher mentioned that this result is important because chordal graphs can be colored in polynomial time [28]. Some researcher implemented various groups code generators integrating optimal instruction selection, instruction scheduling and register allocation, based on formulations such as integer linear programming [29].

3. INSTRUCTION SCHEDULING AND REGISTER ALLOCATION

3.1 Instruction Scheduling

In VLIW ILP compilers schedule various operations on two different functional units. The VLIW compilers perform all the scheduling and translation at compile time. For instruction scheduling, it is assumed that all the functional unit of same kind and have the same latency [30-32]. The VLIW compilers read the program in HLL and translate the complex operations into micro-operations supported by the processor. The next task for the complier is to check the data and control dependencies among the operations and the compiler selects which operations can be executed in parallel.

The conventional algorithms for list scheduling attempts to select the first freely available functional unit to schedule an operation. But in modern commodity of processors, the functional units of same kind may have different latencies. As a result, conventional instruction scheduling algorithm may not produce good performance, at least not if integrated with register allocation [33].

3.2 Register Allocation

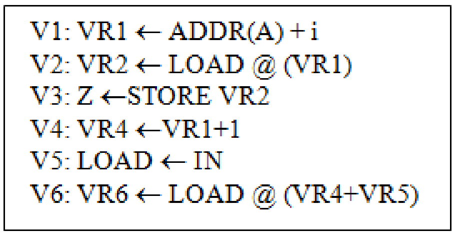
The various schemes for register allocation include: Region based, linear scan, graph coloring, integer programming, SSA-form, etc. The primary job of a register allocator is to assign the many temporals to a small number of CPU registers and to assign the source and destination of a move of same register if possible so that the move can be deleted. In optimizing compiler, the register allocation is the process of assigning large number of program variables on to a limited number of CPU registers. The register allocation can be locally as well as globally. When it is done over a basic block then it is local register allocation and when it is applied to whole function or a program then it is called global register allocation.

A special category called inter-procedural register allocation also exists when register allocation is done across the function boundaries. In trimaran infrastructure, mainly three register allocation schemes are used: IMPACT Register allocation, Linear Scan, and Region Based. This paper opted Linear Scan approach integrated with Fast Instruction Scheduling (FIS). The integrated approach was implemented using subgraph isomorphism.

As an example, a below code is:

```
Z = A(i)
T = A(i+1+N)
```

The intermediate code (basic block) for above source code is:



```
V1: VR1 ← ADDR(A) + i
V2: VR2 ← LOAD @ (VR1)
V3: Z ←STORE VR2
V4: VR4 ←VR1+1
V5: LOAD ← IN
V6: VR6 ← LOAD @ (VR4+VR5)
```

Figure 1: Basic Block

This basic block has six instructions with one two-stage pipeline and two physical registers. Figure 2 is the dependence graph for above basic block (Figure 1):

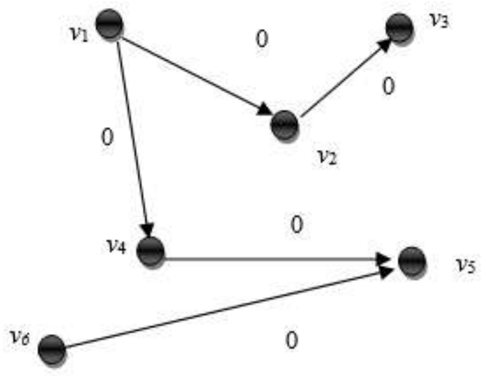


Figure 2: Dependence Graph

The association of instruction scheduling and register allocation has been implemented in three ways: Instruction scheduling followed by register allocation, register allocation followed by instruction scheduling, and integrated Instruction scheduling and Register allocation.

3.2.1 Instruction Scheduling Followed by Register Allocation

In this scheme the priority is given to instruction scheduling over register utilization for exploitation of ILP. In modern RISC processors, if only one approach has to be chosen over instruction scheduling and register allocation, this scheme is preferred. Figure 3 shows the schedule generated by this approach. There is no idle slot and completion is done in 6 cycles.

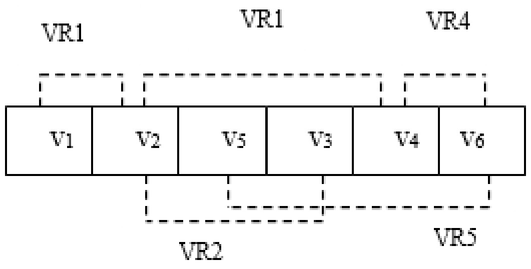


Figure 3: Instruction Scheduling followed by Register Allocation

3.2.2 Register Allocation Followed by Instruction Scheduling

In this scheme the priority is given to register utilization over instruction scheduling for exploitation of ILP. It was most common approach in early compilers but now-a-days it is not used as the sufficient numbers of registers are available because of reduced cost of hardware. Below figure shows the schedule and register allocation for this scheme. There are no spills during register allocation.

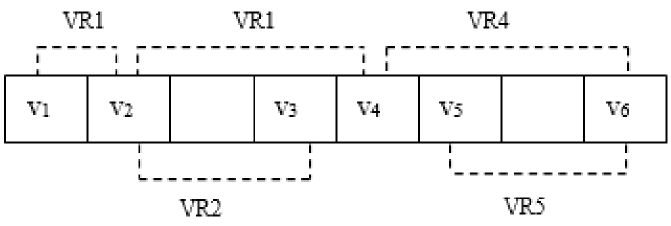


Figure 4: Register Allocation followed by Instruction Scheduling

3.2.3 *Integrated Instruction Scheduling and Register Allocation*

This approach deal with issues related to instruction scheduling and register allocation both. The solution chose to move V5 closer to V6 but does not move V3 closer to V2. Thus, there is one idle slot in the schedule and no spill at all.

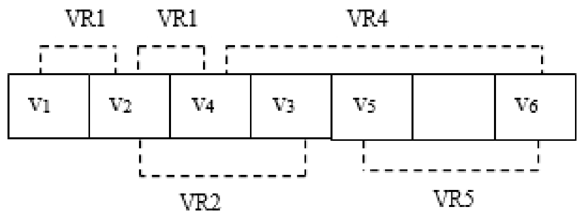


Figure 5: Integrated Instruction scheduling and Register allocation

4. THE HPL-PD ARCHITECTURE

This work is focused on instruction scheduling and register allocation for HPL-PD architecture. HPL-PD is a parametric processor architecture accepted for research in Instruction Level parallelism [31]. The architecture is parametric in the sense that it admits hardware of different specifications and scale exclusively the nature and amount of ILP that can be exploited. HPL-PD implementation provides the merits and demerits of each implementation as well. Figure 6 illustrates an overview of HPL-PD datapath [30]. The code generation for this architecture has been a challenge since there are dedicated functional units associated to the register banks.

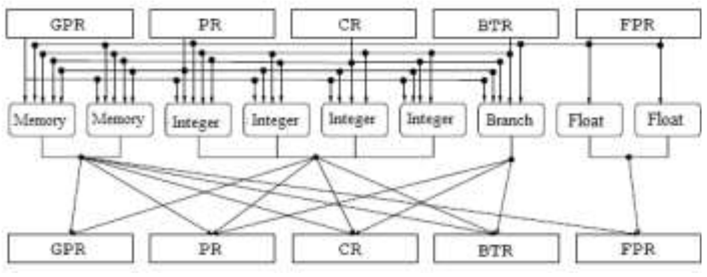


Figure 6: The HPL-PD Datapath

Figure 7 is the Base Graph of HPL-PD processor. In this ABG (Abstract Base Graph), the base files are represented by rectangles and the circles represent the functional units.

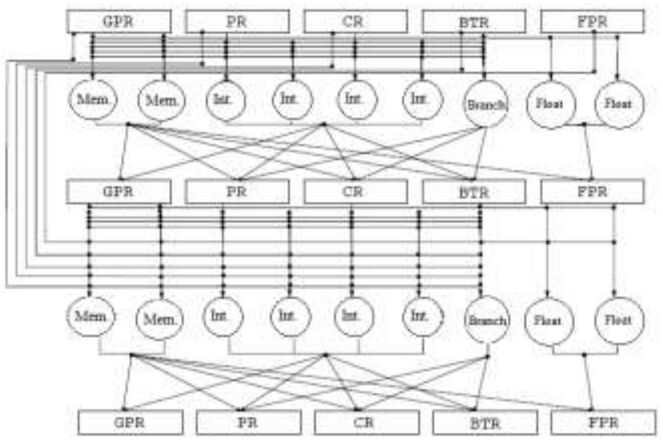


Figure 7: The Base Graph of HPL-PD Processor

HPL-PD version 1.1 is used for experiments. This version has five new integer operations namely: ABS, MIN, MINL, MAX and MAXL. The conversion operations, hold by the architecture are CONVLWS, CONVLWD, CONVLSW and CONVLDW. A number of few MOVE operations are included. Some of them are MOVEGBP, MOVEB, MOVEGCM. HPL-PD is a meta architecture, as a result it encompasses a space of machines each of which can have different amount of ILP and a different ISA (Instruction Set Architecture). Architectures lying in HPL-PD space consists of a set of registers, functional units connected to those registers and a hierarchical memory system.

5. PROPOSED ALGORITHM

The concepts of subgraph isomorphism library is used to design the algorithm and used trimaran infrastructure for implementation of algorithm in association with redefining the machine description [32]. The implementation of this algorithm is attempted as a new pass on the backend.

Algorithm

```
1. Input the program in C language
2. Split the program into basic blocks
3. Construct the DAGs from the basic blocks and base graphs.
4. For each DAG D and base graph B
(a) Compute the unrolling for B
(b) Create unrolled base graph
(c) Perform subgraph matching between D and B
5. If subgraph matching between D and B = false
a) if B is not large enough then  increase the unrolling factor by 1 and goto
4 (b) if there exists a spill then  include vertices representing STORE and
LOAD from memory, and  update DAG D with vertices and goto 4 (c)
c) if matching is not found after 5 (a) and 4 (b) then
  break up the DAG D under matching, and goto 4 (c)
5. Furnish the scheduled and register allocated instructions for each basic
block.
```

Above describes the main steps of proposed algorithm. It takes a C program and converts into basic blocks. Then the algorithm receives the DAG from the basic block. The Base Graph size then calculates the unrolling factor. Larger is the size of unrolling factor, higher is the exploitation of ILP. The unrolling factor is passed to procedure named Graph Creator. Then the subgraph matching is performed. It may be possible that the matching is not found then the subgraph isomorphism runs a specific procedure depending upon the result of matching. If finally, the matching is not found as per the time constraints, the algorithm breaks up the DAG and repeats the matching step. Once the matching is found, the trimaran performs a task of code emission with CPU registers.

For implementation of HPL-PD instruction scheduling and register allocation, the inputs are DAGs, which are generated by Trimaran compiler. An internal procedure builds HPL-PD architecture base graph that accepts DAGs as input and produces ABG.

6. MDES DESCRIPTION STEPS

The MDES (Machine DEScription) model in trimaran infrastructure provides the flexibility to develop a machine description for HPL-PD group of processors in high level language to be translated into equivalent low-level representation used by the compiler at later stage. The purpose of low level representation is to allow the compiler to check the execution constraints efficiently. The HPL-PD machine description is bound to follow a well-defined format called HMDES (High level Machine DEScription).

After the processing of macro and compilation of high level machine description (here, *P. hmdes2*), the corresponding low-level machine description (here, *P. lmdes2*) is loaded to read the LMDES specification and constructs the internal data structures of the MDES database. The information contained within the machine description is made available to various modules of trimaran infrastructure. For optimization of compiler, a machine description database specifies the following to the compiler:

1. A meta grammar
2. An internal data structure for instruction format tree.
3. Explicitly scheduled resources.
4. The resource usage behavior of each operation.
5. Latency description.
6.

   The high-level description is for user's convenience and the compiler performs the activities at low level description. Following script is applied for converting high level description (**. hmdes2*) into low level description (**. lmdes2*):

1. Run the *hc* script /* conversion into *. lmdes2 */
2. Run *hmdesc* /*Generation of customized file for IMPACT user interface to MDES Module. */
3. Processing and compilation of MDES file called by *hc* and *hmdes* scripts by using the binaries *md_processor*, *md_compiler* and *lmdes2_customizer*.

The back-end source files in MDES are integrated in ELCOR. The ELCOR side MDES source is implemented by following steps:

1. Define internal data structure created in mdes. *
2. Define the function for loading **.lmdes2* file in *mdes_reader.cpp*
3. Include the object files of the ELCOR side of the MDES library *libmdes.a*

The front end side MDES source is integrated in *libmspec.a* library that contains the object modules. The directory *TRIMARAN_HOME/impact/src/machine* contains all the files related to high level machine description.

7. EXPERIMENTAL RESULTS

In this section the summery of experiments is presented. The experimental setup is prepared as integrated approach for instruction scheduling and register allocation. The table below shows the number of emitted instructions and registers usages for each benchmark. The experiments performed using the basic and greedy registers allocators and fast scheduling algorithm for trimaran compiler (version 4.0) on Ubuntu 10.10.

Table 1: Summary of Instruction Scheduling and Register Allocation Experiments

Name of Benchmark	Emitted Instructions	Allocated Registers
alloca_test	42	09
bmm	65	12
dag	38	07
eight	43	09
fact2	28	04
fft	735	122
fib	49	08
fib_mem	36	13
fir	73	16
fir_int	51	11
hyper	37	07
ifthen	69	22
local_var_test	46	13
longlong	32	08
mm	112	35
mm_double	52	11
mm_int	43	09
nested_complex	285	64
sqrt	48	10
strcpy	50	09
struct_test	69	24
switch_test	612	96
type_test	142	23
wave	38	08

8. COMPARISON OF RESULTS WITH LLVM COMPILER ON MIPS ARCHITECTURE

Figure 7 shows the comparison of emitted instruction between Trimaran and LLVM. The experiments show that the emitted instruction by trimaran on HPL-PD produce better results than the LLVM on MIPS. The number of emitted instructions are more in Trimaran than LLVM. The scheduler performance gain of proposed approach can be observed for all the benchmarks used for Trimaran on HPL-PD than the LLVM on MIPS.

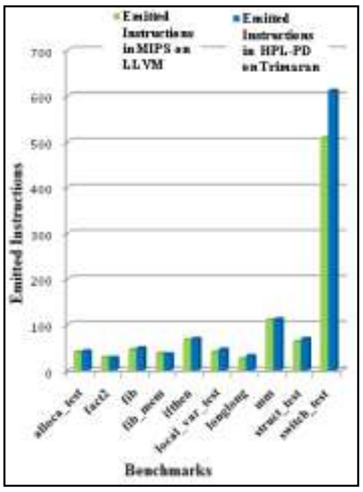


Figure 7: Comparison of Emitted Instructions

The figure 8 shows the register usages for both the compilers. Here observations are that less number of registers used by HPL-PD than MIPS. The integrated approach for Trimaran using subgraph isomorphism leads to fewer registers for all the evaluated benchmarks.

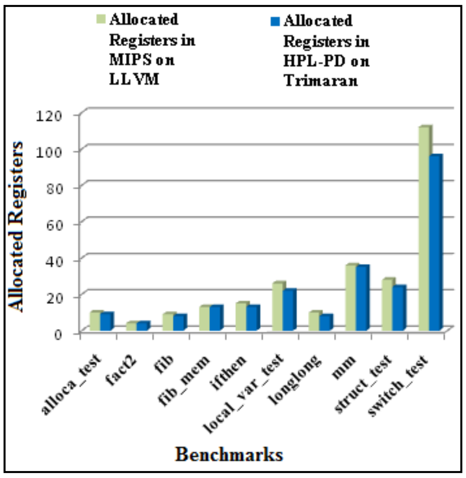


Figure 8: Comparison of Register Utilization

9. CONCLUSION AND FUTURE SCOPE

The comparison of integrated approach for instruction scheduling and register allocation between LLVM and trimaran compilers has been presented in this paper. The comparison was based on matching the DAGs to the base graph on LLVM and Trimaran compiler. The matching result is in the form of subgraph of the base graph isomorphic to the input DAG that represents the allocated resources to run the DAG. In this paper it is shown that proposed algorithm allocates fewer registers per benchmark and spills fewer temporaries. The fast and basic strategy provided better results than isomorphism strategy lying in the range 05 ms – 20 ms. The average compilation time achieved was 49 ms for isomorphism and 37 ms for fast and basic strategy. On average spill codes generated for the all benchmarks were 0.03% dynamically and 0.05% statically.

The Intel/Itanium processors are HPL-PD based. The optimization and analysis models can be improved with HPL-PD processors. HPL-PD configurations can widely be used in computer architecture research. It can provide ideal simulation environment for machine learning. As future work, some new parameters can be included for analysis for base graph heuristic. Evaluation of the scheduling algorithm may be introduced under the architectures based on multiple processing elements dynamic schemes.

REFERENCES

[1] Santos, L.S.R., Silva, R. 2012. An Integrated Technique for Instruction Scheduling and Register Allocation Based on Subgraph Isomorphism. Proceedings of the 16th Brazilian Symposium on Programming Languages, Brazil, 1-5.

[2] Lozano, R.C., Carlsson, M., Drejhammar, F., Schulte, C. 2012. Constraint-Based Register Allocation and Instruction Scheduling. Conference Proceeding, Springer-Verlag Berlin Heidelberg, LNCS, 7514, 750 –766.

[3] Tang, H., Yang, X., Wang, S., Zhang, Y. 2013. Optimizing Instruction Scheduling and Register Allocation for Register-File-Connected Clustered VLIW Architectures. The Scientific World Journal, 1-11.

[4] Chakrapani, L.N., Gyllenhaal, J.C., Hwu, W.W., Mahlke, S.A., Palem, K.V., Rabbah, R.M. 2005. Trimaran: An Infrastructure for Research in Instruction-Level Parallelism. *LCPC* 2005, 32-41.

[5] Kumar, M., Mishra, S. 2014. Approximation Algorithms for Node Deletion Problems on Bipartite Graphs with Finite Forbidden Subgraph Characterization. Journal of Theoretical Computer Science, 526, 90-96.

[6] Kathail, V., Schlansker, M.S., Rau, B.R. 2000. HPL-PD Architecture Specification: Version 1.1, Technical report, HP Laboratories Palo Alto, HPL-93-80 (R.1).

[7] Chow, P. 1988. MIPS-X Instruction Set and Programmer's Manual. Technical Report, Natural Sciences and Engineering Research Council of

Canada, No. CSL-86-289.

[8] Lattner, C., Adve, V. 2004. The LLVM Compiler Framework and Infrastructure Tutorial. Proceedings of the 17th international conference on Languages and Compilers for High Performance Computing, USA, 15-16.

[9] Magno, F., Pereira, Q. 2014. A survey on Register Allocation. US Patent 8,732,680 B2, May 20.

[10] Chaitin, G.J., Auslander, M.A., Chandra, A.K., Cocke, J., Hopkins, M.E., Markstein, P.W. 1981. Register Allocation via Coloring. ACM Journal Computer languages, 6, 47 -57.

[11] Sarkar, Barik. 2007. Extended Linear Scan: An Alternate Foundation for Global Register Allocation. ACM, Proceeding LCTES/CC, 141-148.

[12] Björklund, A., Husfeldt, T. 2008. Exact Graph Coloring Using Inclusion–Exclusion. Encyclopedia of Algorithms, Springer US, 289.

[13] Karp, R. 1972. Reducibility among combinatorial problems. Complexity of Computer Computations, Plenum, New York, 85-103.

[14] Higuera, D.L., Janodet, C., Samuel, J.C., Émilie, Damiand, Guillaume, Solnon, Christine. 2013. Polynomial algorithms for open plane graph and subgraph isomorphisms. Theoretical Computer Science, 76–99.

[15] Wang, S., Zhang, S., Yang, Y. 2014. Hamiltonian Path Embeddings in Conditional Faulty k-ary n-cubes. Journal of Information Sciences, 268, 463-488.

[16] Elkin, M. 2005. Computing Almost Shortest Paths. ACM Transactions on Algorithms, 1 (2), 283-323.

[17] Cheng, J., Ke, Y., Fu, A.W.C., Yu, J.X., Zhu, L. 2011. Finding Maximal cliques in Massive Networks. ACM Transactions on Database Systems, 36 (4), 1–21.

[18] Hack, S., Grund, D., Goos, G. 2006. Register allocation for programs in SSA-Form. Proceedings of the 15th International Conference Theory and Practice of Software, ETAPS, Vienna, Austria, 247-262.

[19] Braun, M., Hack, S. 2009. Register Spilling and Live-Range Splitting for SSA-Form Programs. 18th International Conference, CC 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, York, UK, 174–189.

[20] Kumar, R., Singh, P.K. 2014. An Approach for Compiler Optimization to Exploit Instruction Level Parallelism. Proceeding of International Conference ICACNI-14 (Springer), Kolkata, 509-16.

[21] Nagarajan, R., Kushwaha, S.K., Burger, D., McKinley, K.S., Lin, C., Keckler, S. 2004. Static Placement, Dynamic Issue (SPDI) Scheduling for EDGE Architectures. Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, IEEE Computer Society Washington, 74–84.

[22] Tobias, J.K., Koch, E.V., Bohm, I., Franke, B. 2010. Integrated Instruction Selection and Register Allocation for Compact Code Generation Exploiting Freeform Mixing of 16- and 32-bit Instructions. Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization, New York, USA, 180–189.

[23] Koes, D., Goldstein, S.C. 2005. A Progressive Register Allocator for Irregular Architectures. Proceeding of International Symposium on Code Generation and Optimization (CGO'05), Washington, 269–280.

[24] Bouchez, F., Darte, A., Rastello, F. 2007. On the complexity of spill everywhere under SSA form. Proceedings of the ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, NY, USA, 42 (7), 103 – 112.

[25] Pereira, F.M.Q., Palsberg, J. 2005. Register Allocation Via Coloring of Chordal Graphs. Programming Languages and Systems, Lecture Notes in Computer Science, 3780, 315-329.

[26] Brisk, P., Dabiri, F., Macbeth, J., Sarrafzadeh, M. 2005. Polynomial-Time Graph Coloring Register Allocation. 14th International Workshop on Logic and Synthesis, Lake Arrowhead, California.

[27] Hack, S., Grund, D., Goos, G. 2006. Register allocation for programs in SSA-form. Proceeding of 15th International Conference as Part of the Joint European Conferences on Theory and Practice of Software, Vienna, Austria, 247–262.

[28] Gavril, F. 1972. Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph. SIAM Journal of Computing, 1, 180 – 187.

[29] Eriksson, M.V., Skoog, O., Kessler, C.W. 2008. Optimal vs. Heuristic Integrated Code Generation for Clustered VLIW Architectures. Proceedings of the 11th International Workshop on Software & Compilers for Embedded Systems, New York, USA, 11-20.

[30] Schlansker, M.S., Rau, B.R. 2000. EPIC: Explicitly Parallel Instruction Computing. IEEE Journal of Computer, 33 (2), 37 – 45.

[31] Rajendra, K., Singh, P.K. 2010. A Modern Parallel Register Sharing Architecture for Code Compilation. International Journal of Computer Applications, 1 (16), 95-99.

[32] Blankstein, A., Goldstein, M. 2010. Subgraph Isomorphism, Technical Report, MIT 6.884.

[33] Fu, H., Liao, J., Yang, J., Wang, L., Song, Z., Huang, X., Yang, C., Xue, W., Liu, F., Qiao, F., Zhao, W., Yin, X., Hou, C., Zhang, C., Ge, W., Zhang, J., Wang, Y., Zhou, C., Yang, G. 2016. The Sunway Taihu Light supercomputer: system and applications, Science China Information Sciences, 59 (7), 1–16.