Appendix A
List of Techniques

The list starts on the next page, and its legend appears at the end of the list. The techniques are ordered chronologically.

Note that the capabilities of all techniques have been set to reflect those exhibited by current implementation prototypes and known applications, not the capabilities that could potentially be achieved through extensions of the technique.

© Springer International Publishing Switzerland 2016

G. Hjort Blindell, *Instruction Selection*,
DOI 10.1007/978-3-319-34019-7

REFERENCES	PR	SC	OP	MO	DO	IB	IN	KNOWN APPLICATIONS
Lowry and Medlock [243]	ME	L	○	○	○	○	○	FHC
Orgass and Waite [273]	ME	L	○	○	○	○	○	SIMCMP
Elson and Rake [108]	ME	L	○	○	○	○	○	
Miller [255]	ME	L	○	○	○	○	○	DMACS
Wilcox [338]	ME	L	○	○	○	○	○	
Wasilew [331]	TC	L	○	○	○	○	○	
Donegan [101]	ME	L	○	○	○	○	○	
Tirrell [319]	ME	L	○	○	○	○	○	
Weingart [332]	TC	L	○	○	○	○	○	
Ammann et al. [12, 13]	ME	L	○	○	○	○	○	
Young [350]	ME	L	○	○	○	○	○	
Newcomer [263]	TC	L	○	○	○	○	○	
Simoneaux [309]	ME	L	○	○	○	○	○	
Snyder [310]	ME	L	○	○	○	○	○	
Fraser [140, 141]	ME	L	○	○	○	○	○	
Ripken [294]	TC	L	●	○	○	○	○	
Glanville and Graham [158]	TC	L	○	○	○	○	○	
Johnson [191, 192]	TC	L	○	○	○	○	○	PCC
Harrison [170]	ME$^+$	L	○	○	○	○	○	
Cattell [67, 70, 234]	TC	L	○	○	○	○	○	PQCC
Auslander and Hopkins [33]	ME$^+$	L	○	○	○	○	○	
Ganapathi and Fischer [146, 147, 148, 149]	TC	L	○	○	○	○	○	
Krumme and Ackley [218]	ME	L	○	○	○	○	○	
Deutsch and Schiffman [96]	ME	L	○	○	○	○	○	SMALLTALK-80
Christopher et al. [76]	TC	L	●	○	○	○	○	
Davidson and Fraser [91]	ME$^+$	L	○	●	●	○	○	GCC, ACK, ZEPHYR/VPO
Henry [177]	TC	L	●	○	○	○	○	
Aho et al. [6, 7, 321]	TC	L	●	○	○	○	○	TWIG
Hatcher and Christopher [172]	TC	L	●	○	○	○	○	
Horspool [184]	TC	L	●	○	○	○	○	
Fraser and Wendt [135]	ME$^+$	L	○	○	○	○	○	
Giegerich and Schmal [157]	TC	L	○	○	○	○	○	

References	PR	SC	OP	MO	DO	IB	IN	Known applications
Hatcher and Tuller [174]	TC	L	●	○	○	○	○	UNH-Codegen
Pelegrí-Llopart and Graham [278]	TC	L	●	○	○	○	○	
Yates and Schwartz [348]	TC	L	●	○	○	○	○	
Emmelmann et al. [109]	ME	L	●	○	○	○	○	BEG, CoSy
Ganapathi [145]	TC	L	○	●	○	○	○	
Genin et al. [155]	ME$^+$	L	○	●	○	○	○	
Nowak and Marwedel [269]	DC	L	○	●	●	○	○	MSSC
Balachandran et al. [35]	TC	L	●	○	○	○	○	
Despland et al. [63, 94, 95]	TC	L	○	○	○	○	○	Pagode
Wendt [335]	ME$^+$	L	○	●	○	○	○	
Hatcher [173]	TC	L	●	○	○	○	○	UCG
Fraser et al. [137]	TC	L	●	○	○	○	○	IBurg, Record, Redaco
Fraser et al. [138, 281, 282, 284, 285]	TC	L	●	○	○	○	○	Burg, HBurg, JBurg, WBurg, OCamlBurg
Emmelmann [110]	TC	L	○	○	○	○	○	
Wess [336, 337]	TD	L	●	○	○	○	○	
Marwedel [252]	DC	L	○	●	●	○	○	MSSV
Tjiang [320]	TC	L	●	○	○	○	○	Olive, Spam
Engler and Proebsting [114]	TC	L	●	○	○	○	○	DCG
Fauth et al. [125, 258]	DC	L	○	●	○	○	○	CBC
Ferdinand et al. [127]	TC	L	●	○	○	○	○	
Liem et al. [237, 276, 277]	DC	L	○	○	○	○	○	CodeSyn
Lanneer et al. [222, 326, 327]	GC	G	●	●	●	○	○	Chess
Wilson et al. [340]	DC	L	●	●	○	○	○	
Yu and Hu [351, 352]	DC	L	●	●	○	○	○	
Novack et al. [267, 268]	MS	G	●	●	○	○	○	
Hanson and Fraser [169]	TC	L	●	○	○	○	○	LBurg, LCC
Liao et al. [235, 236]	DC	L	●	●	○	○	○	
Adl-Tabatabai et al. [1]	ME	L	○	○	○	○	○	Omniware
Engler [113]	ME	L	○	○	○	○	○	VCode
Hoover and Zadeck [182]	DC	L	○	●	●	○	○	
Leupers and Marwedel [226, 233]	DC	L	○	●	●	○	○	
Nymeyer et al. [270, 271]	TC	L	○	○	○	○	○	

References	PR	SC	OP	MO	DO	IB	IN	Known applications
Shu et al. [308]	TC	L	○	○	○	○	○	
Gough [160, 161, 162]	TC	L	●	○	○	○	○	MBurg, GPBurg
Gebotys [152]	DC	L	●	●	○	○	○	
Hanono and Devadas [167, 168]	TD	L	○	○	○	○	○	Aviv
Leupers and Marwedel [230]	DC	L	○	●	●	○	○	MSSQ
Bashford and Leupers [40]	DC	L	●	●	●	○	○	
Ertl [117]	DC	L	●	●	○	○	○	DBurg
Fraser and Proebsting [139]	ME	L	○	○	○	○	○	GBurg
Fröhlich et al. [142]	TD	L	●	○	○	○	○	
Visser [328]	GC	G	○	●	●	○	○	
Leupers [228]	DC	L	○	●	●	○	○	
Madhavan et al. [245]	TC	L	●	○	○	○	○	
Arnold and Corporaal [25, 26, 27]	DC	L	○	●	○	○	○	
Sarkar et al. [300]	DC	L	○	○	○	○	○	Jalapeño
Paleczny et al. [274]	GC	G	●	○	○	○	○	JHSC
Lorenz et al. [241, 242]	DC	L	○	●	○	○	○	
Bravenboer and Visser [56]	TC	L	●	○	○	○	○	
Krishnaswamy and Gupta [216]	ME$^+$	L	○	●	●	○	○	
Eckstein et al. [106]	GC	G	○	○	○	○	○	
Tanaka et al. [317]	DC	L	○	●	●	○	○	
Borchardt [50]	TC	L	●	○	○	○	○	
Brisk et al. [58]	DC	L	○	●	○	○	○	
Cong et al. [82]	DC	L	○	●	●	○	○	
Lattner and Adve [224]	DC	L	○	○	○	○	○	LLVM
Kessler et al. [43, 115, 116]	DC	L	●	●	○	○	○	Optimist
Clark et al. [77]	DC	L	○	●	●	○	○	
Dias and Ramsey [98]	ME$^+$	L	○	●	○	○	○	
Ertl et al. [118]	TC	L	●	○	○	○	○	
Farfeleder et al. [121]	TC	L	○	◐	○	○	○	
Kulkarni et al. [219]	ME$^+$	L	○	●	●	○	○	Vista
Hormati et al. [183]	DC	L	○	●	●	○	○	
Scharwaechter et al. [303]	DC	L	○	○	●	○	○	CBurg

References	PR	SC	OP	MO	DO	IB	IN	Known applications
Ebner et al. [105]	GC	G	○	●	●	○	○	
Koes and Goldstein [212]	DC	L	○	●	○	○	○	NOLTIS
Ahn et al. [2]	DC	L	○	●	●	○	○	
Martin et al. [248, 249]	DC	L	●	●	●	○	○	
Buchwald and Zwinkau [61]	GC	G	○	○	○	○	○	
Dias and Ramsey [97, 289]	ME$^+$	L	○	●	○	○	○	
Edler von Koch et al. [107]	TC	L	○	○	○	○	○	
Floch et al. [131]	DC	L	●	●	●	○	○	
Yang [347]	TC	L	●	○	○	○	○	
Youn et al. [349]	DC	L	○	◐	●	○	○	CBURG
Arslan and Kuchcinski [29, 30]	DC	L	●	●	●	○	○	
Janoušek and Málek [187]	TC	L	○	○	○	○	○	
Andrade [16]	ME	L	○	○	○	○	○	GNU LIGHTNING

Fundamental principle (PR) on which the technique is based: macro expansion (ME), macro expansion with peephole optimization (ME$^+$), tree covering (TC), trellis diagrams (TD—is actually sorted under TC in this book), DAG covering (DC), graph covering (GC), or mutation scheduling (MS—is actually sorted under GC in this book). Scope of instruction selection (SC): local (L, isolated to a single block), or global (G, considers entire functions). Whether the technique is claimed to be optimal (OP). Supported machine instruction characteristics: single-output (supported by all techniques), multi-output (MO), disjoint-output (DO), inter-block (IB), and interdependent (IN) instructions. The symbols ○, ◐, and ● indicate no, partial, and full support, respectively.

Appendix B
Publication Timeline

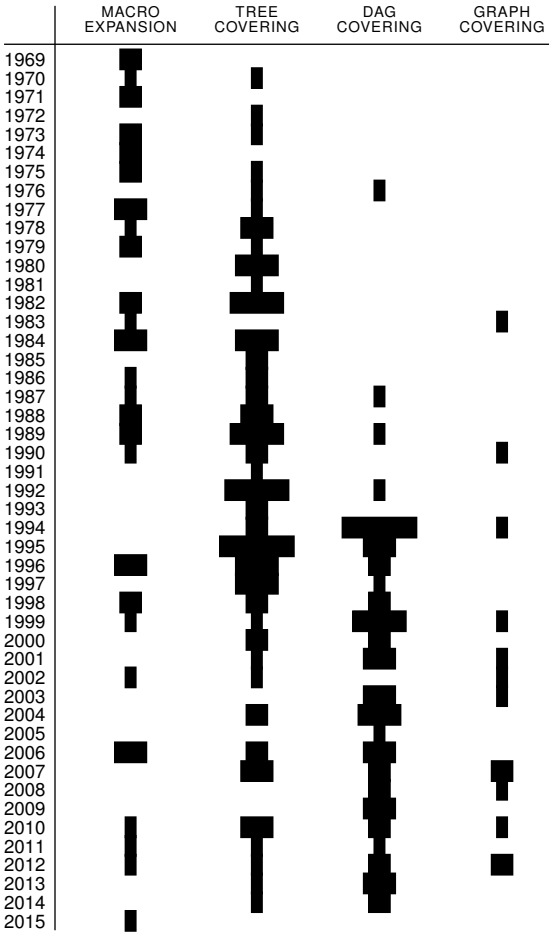

Fig. B.1: Illustrates how research on instruction selection (201 publications in total), with respect to the fundamental principles, has elapsed over time. The widths of the bars indicate the relative numbers of publications for a given year

© Springer International Publishing Switzerland 2016

G. Hjort Blindell, *Instruction Selection*,
DOI 10.1007/978-3-319-34019-7

Appendix C
Graph Definitions

A *graph* is defined as a tuple (N,E) where N is a set of *nodes* (also known as *vertices*) and E is a set of *edges*, each consisting of a pair of nodes $n,m \in N$. A graph is *undirected* if its edges have no direction, and *directed* if they do. In a directed graph we write an edge from a node n to another node m as $n \to m$, and say that such an edge is *outgoing* with respect to n, and *ingoing* with respect to m. We also introduce the following functions:

$$src : E \to N \qquad dst : E \to N$$
$$src(n \to m) = n \;\; dst(n \to m) = m$$

Edges for which $src(e) = dst(e)$ are known as *loop edges* (or simply *loops*). If there exists more than one edge between the same pair of nodes then the graph is a *multigraph*, otherwise it is a *simple* graph.

A list of edges that describe how to get from one node to another is called a *path*. More formally we define a path between two nodes n and m as an ordered list of edges $p = \langle e_1, \ldots, e_l \rangle$ such that for the directed graph (N,E):

$$e_i \in E \;\; \forall e_i \in p$$
$$dst(e_i) = src(e_{i+1}) \;\; \forall 1 \leq i < l-1$$

Paths for undirected graphs are similarly defined and will thus be skipped. A path for which $src(e_1) = dst(e_l)$ is known as a *cycle*. Two nodes n and m, where $n \neq m$, are said to be *connected* if there exists a path from n to m, and if the path is of length 1 then n and m are also *adjacent*. A directed graph containing no cycles is known as a *directed acyclic graph (DAG)*. An undirected graph is *connected* if and only if there exists a path for every distinct pair of nodes. For completeness, a directed graph is *strongly connected* iff, for every pair of distinct nodes n and m, there exists a path from n to m and a path from m to n. Also, a directed graph is *weakly connected* if replacing all edges with undirected edges yields a connected undirected graph. An example is shown in Fig. C.1.

© Springer International Publishing Switzerland 2016

G. Hjort Blindell, *Instruction Selection*,
DOI 10.1007/978-3-319-34019-7

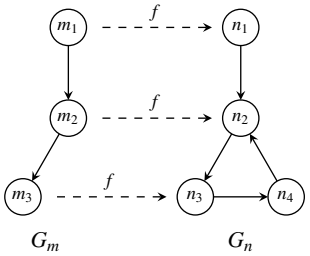

Fig. C.1: An example of two simple directed graphs G_m and G_n. Through the graph homomorphism f we see that G_m is an isomorphic subgraph of G_n. Both graphs are weakly connected, and G_n also has a strongly connected subgraph, consisting of n_2, n_3, and n_4 which form a cycle

A simple, undirected graph that is connected, contains no cycles, and has exactly one path between any two nodes is called a *tree*. A set of trees constitutes a *forest*. Nodes in a tree with exactly one neighbor are known as *leaves*. A *directed tree* is a directed graph that would become a tree when ignoring the direction of its edges. A *rooted directed tree* is a directed tree where one node has been assigned the *root* and all edges either point away or towards the root. In a rooted directed tree a *parent* of a node n is the node adjacent to n that is closest to the root. Likewise, if a node n is the parent of another node m, then m is a *child* of n. In this book we assume all trees to be rooted directed trees, and a tree will always be drawn with its root appearing at the top.

A graph $G = (N, E)$ is a *subgraph* of another graph $G' = (N', E')$, also written as $G \subseteq G'$, iff $N \subseteq N'$ and $E \subseteq E'$. Likewise, a tree T is a *subtree* of another tree T' iff $T \subseteq T'$.

A *graph homomorphism* is a mapping between two graphs such that their structure is preserved. More formally, a graph homomorphism f from a graph $G = (N, E)$ to another graph $G' = (N', E')$ is a mapping $f : N \to N'$ such that $(u, v) \in E$ implies $(f(u), f(v)) \in E'$. If the graph homomorphism f is an injective function, then f is also called a *subgraph isomorphism*. If there exists such a mapping then we say that G is an *isomorphic subgraph* of G', and an example of this is given in Fig. C.1. If f is a bijection, whose inverse function is also a graph homomorphism, then f is also called a *graph isomorphism*.

Lastly we introduce the notion of *topological sort*, where the nodes of a graph (N, E) are arranged in an ordered list $\langle n_1, \ldots, n_n \rangle$ such that $n_i \in N \ \forall 1 \leq i \leq n$ and for no pair of nodes n_i and n_j, where $i < j$, does there exist an edge $n_j \to n_i \in E$. In other words, if the edges are added to the list then all edges will go point forward from left to right (hence topological sorts are only defined for DAGs). Several methods exists for achieving a topological sort, see for example Section 22.4 in Cormen et al. [88].

Appendix D
Taxonomy

Technical terms and and their exact definitions often differ from one publication to another, thus making it difficult to discuss and compare techniques without a common foundation. In this book, therefore, a taxonomy with a well-defined vocabulary has been established and is used consistently throughout the book. Although this may seem a bit daunting and superfluous at first, most items in the taxonomy are easy to understand. Most importantly, having explicitly defined these terms will minimize confusions that may otherwise occur.

D.1 Common Terms

Several terms are continuously used when discussing the instruction selection techniques. Most of these are obvious and appear in other literature, while others may require a little explanation.

Program. The code under compilation, and therefore the input to the compiler as well as the instruction selector. In the former this refers to the source code, while in the latter it usually refers to the intermediate representation (IR) code, either in its entirety or parts of it (that is, a function, a basic block, or part of a block), depending on the scope of the instruction selector.

Target machine. The hardware on which the program is compiled to run. Most often this refers to the ISA implemented and understood by its processing unit.

Instruction selector. A component or program responsible for implementing and executing the task of instruction selection. If this program is automatically generated from a machine description, the term refers to the generated *result* and not the *generator* itself.

Frontend. A component or program responsible for parsing, validating, and translating the program into equivalent IR code.

Code generation. The task of generating assembly code for a given program by performing instruction selection, instruction scheduling, and register allocation.

© Springer International Publishing Switzerland 2016

G. Hjort Blindell, *Instruction Selection*,
DOI 10.1007/978-3-319-34019-7

Backend. A component or program responsible for implementing and executing the task of code generation.

Compilation time. The time required to compile a given program.

Pattern matching. The problem of detecting when and where it is possible to use a certain instruction for a given program.

Pattern selection. The problem of deciding which instructions to select from the set of candidates found during pattern matching.

Offline cost analysis. The task of shifting the computation of optimization decisions from the phase of program compilation to the phase of compiler generation, thereby reducing compilation time at the cost of increasing the time it takes to generate the compiler.

D.2 Machine Instruction Characteristics

A machine instruction exhibits one or more machine instruction characteristics. For this study, the following characteristics were identified:

Single-output instructions. An instruction that only produces a single observable output value. In this context observable means a value that can be accessed by the program. This includes instructions that perform typical arithmetic operations such as addition and multiplication as well as bit operations, but it also includes instructions that chain multiple computations (for example, "load into register $\texttt{r}_\texttt{d}$ the value at memory location specified in register $\texttt{r}_\texttt{x}$ plus offset specified in register $\texttt{r}_\texttt{y}$ plus an immediate value"). Note that the instruction must produce only this value and nothing else (compare this with the next characteristic).

Multi-output instructions. An instruction that produces multiple observable output values from the same input values. Examples include `divmod` instructions that produce both the quotient as well as the remainder of two terms, but it also includes instructions that set status flags in addition to computing the arithmetic result. A status flag is a bit that signifies additional information about the result (for example, if there was a carry overflow or the result was equal to 0) and is therefore often a side effect of the instruction. In reality, however, these bits are nothing else but additional output values produced by the instruction.

Disjoint-output instructions. An instruction that produces multiple observable output values from disjoint input value sets. This means that if the expression for computing each observable output value formed a graph, then these graphs would be disjoint. In comparison, single-output and multi-output instructions all form a single graph. Disjoint-output instructions typically include SIMD instructions which execute the same operations simultaneously on many input values.

Inter-block instructions. An instruction whose execution essentially spans multiple blocks. Examples of such instructions are saturated arithmetic instructions and hardware-loop instructions, which repeat a fixed sequence of instructions a certain number of times. As seen in this book, no instruction selection design is yet capable of supporting this kind of instruction.

Interdependent instructions. An instruction that enforces additional constraints when appearing in combination with other instructions in the assembly code. An example is the `add` instruction from the TMS320C55x instruction set which cannot be combined with an `rpt k` instruction if the addressing mode is set to a specific mode. This kind of instruction is very difficult to handle by most instruction selection techniques.

The first three characteristics form sets of instructions that are disjoint from one another, but the last two characteristics can be combined as appropriate with any of the other characteristics. For example, the same instruction can exhibit single-output, inter-block, as well as the characteristics of interdependent instructions.

D.3 Scope

Local instruction selection. Selects instructions for a single block at a time.

Global instruction selection. Selects instructions for several blocks or an entire function at a time.

D.4 Principles

All techniques reviewed in this book have been categorized into one of four principles.

Macro expansion. Each IR node in the program is expanded into one or more instructions using macros. This is a simple strategy but generally produces very inefficient assembly code as an instruction often can implement more than one IR node. Consequently modern instruction selectors that apply this approach also incorporate peephole optimization that combines many instructions into single equivalents.

Tree covering. The program and instructions are represented as trees. Each instruction gives rise to a pattern tree which is then matched over the program tree (this is the pattern matching problem). From the matching set of patterns, a subset is selected such that the entire program tree is covered at the lowest cost.

DAG covering. The same idea as tree covering but operates on DAGs instead of trees. Since DAGs are a more general form of trees, DAG covering supersedes tree covering.

Graph covering. The same idea as DAG covering but operates on general graphs instead of DAGs. Since graphs are a more general form of DAGs, graph covering supersedes DAG covering.

References

[1] A.-R. Adl-Tabatabai, G. Langdale, S. Lucco, and R. Wahbe. “Efficient and Language-Independent Mobile Programs”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. SIGPLAN’96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 127–136. ISBN: 0-89791-795-2

[2] M. Ahn, J. M. Youn, Y. Choi, D. Cho, and Y. Paek. “Iterative Algorithm for Compound Instruction Selection with Register Coalescing”. In: *Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*. DSD’09. Washington, District of Colombia, USA: IEEE Computer Society, 2009, pp. 513–520. ISBN: 978-0-7695-37825

[3] A. V. Aho and S. C. Johnson. “Optimal Code Generation for Expression Trees”. In: *Journal of the ACM* 23.3 (1976), pp. 488–501. ISSN: 0004-5411

[4] A. V. Aho, S. C. Johnson, and J. D. Ullman. “Code Generation for Expressions with Common Subexpressions”. In: *Proceedings of the 3rd SIGACT-SIGPLAN Symposium on Principles on Programming Languages*. POPL’76. Atlanta, Georgia, USA: ACM, 1976, pp. 19–31

[5] A. V. Aho and M. J. Corasick. “Efficient String Matching: An Aid to Bibliographic Search”. In: *Communications of the ACM* 18.6 (1975), pp. 333–340. ISSN: 0001-0782

[6] A. V. Aho and M. Ganapathi. “Efficient Tree Pattern Matching: An Aid to Code Generation”. In: *Proceedings of the 12th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’85. New Orleans, Louisiana, USA: ACM, 1985, pp. 334–340. ISBN: 0-89791-147-4

[7] A. V. Aho, M. Ganapathi, and S. W. K. Tjiang. “Code Generation Using Tree Matching and Dynamic Programming”. In: *Transactions on Programming Languages and Systems* 11.4 (1989), pp. 491–516. ISSN: 0164-0925

[8] V. A. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd ed. Boston, Massachusetts, USA: Addison-Wesley, 2006. ISBN: 978-0321486813

© Springer International Publishing Switzerland 2016

G. Hjort Blindell, *Instruction Selection*,
DOI 10.1007/978-3-319-34019-7

[9] P. Aigrain, S. L. Graham, R. R. Henry, M. K. McKusick, and E. Pelegrí-Llopart. “Experience with a Graham-Glanville Style Code Generator”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’84. Montreal, Canada: ACM, 1984, pp. 13–24. ISBN: 0-89791-139-3

[10] R. Allen and K. Kennedy. “Automatic Translation of FORTRAN Programs to Vector Form”. In: *Transactions on Programming Language Systems* 9.4 (1987), pp. 491–542. ISSN: 0164-0925

[11] O. Almer, R. Bennett, I. Böhm, A. Murray, X. Qu, M. Zuluaga, B. Franke, and N. Topham. “An End-to-End Design Flow for Automated Instruction Set Extension and Complex Instruction Selection based on GCC”. In: *1st International Workshop on GCC Research Opportunities*. GROW’09. Paphos, Cyprus, 2009, pp. 49–60

[12] U. Ammann, K. V. Nori, K. Jensen, and H. Nägeli. *The PASCAL (P) Compiler Implementation Notes*. Tech. rep. Eidgenössische Technishe Hochschule, Zürich, Switzerland: Instituts für Informatik, 1974

[13] U. Ammann. *On Code Generation in a PASCAL Compiler*. Tech. rep. Eidgenössische Technishe Hochschule, Zürich, Switzerland: Instituts für Informatik, 1977

[14] B. Anckaert, B. Sutter, D. Chanet, and K. Bosschere. “Steganography for Executables and Code Transformation Signatures”. In: *Proceedings of the 7th International Conference on Information and Communications Security (ICICS’05)*. Ed. by C. Park and S. Chee. Vol. 3506. Lecture Notes in Computer Science. Springer, 2005, pp. 425–439. ISBN: 978-3-540-26226-8

[15] J. P. Anderson. “A Note on Some Compiling Algorithms”. In: *Communications of the ACM* 7.3 (1964), pp. 149–150. ISSN: 0001-0782

[16] P. Andrade. *GNU lightning*. 2015. URL: `http://www.gnu.org/software/lightning/` (visited on 2015-06-03)

[17] Anonymous reviewer. *Private feedback*. 2015

[18] A. W. Appel and J. Palsberg. *Modern Compiler Implementation in Java*. 2nd ed. Cambridge, England: Cambridge University Press, 2002. ISBN: 0-521-82060-X

[19] A. Appel, J. Davidson, and N. Ramsey. *The Zephyr Compiler Infrastructure*. Tech. rep. Charlottesville, Virginia, USA: University of Virginia, 1998

[20] P. Arató, S. Juhász, Z. A. Mann, A. Orbán, and D. Papp. “Hardware-Software Partitioning in Embedded System Design”. In: *International Symposium on Intelligent Signal Processing*. Washington, District of Colombia, USA: IEEE Computer Society, Sept. 2003, pp. 197–202

[21] G. Araujo and S. Malik. “Optimal Code Generation for Embedded Memory Non-Homogeneous Register Architectures”. In: *Proceedings of the 8th International Symposium on System Synthesis*. ISSS’95. Cannes, France: ACM, 1995, pp. 36–41. ISBN: 0-89791-771-5

[22] G. Araujo, S. Malik, and M. T.-C. Lee. “Using Register-Transfer Paths in Code Generation for Heterogeneous Memory-Register Architectures”. In: *Proceedings of the 33rd Annual Design Automation Conference*. DAC’96. Las Vegas, Nevada, USA: ACM, 1996, pp. 591–596. ISBN: 0-89791-779-0

[23] *ARM11 MPCore Processor*. ARM DDI 0360F. Version r2p0. ARM. Oct. 15, 2018

[24] *ARM Cortex-M7 Devices: Generic User Guide*. ARM DUI 0646A. ARM. Mar. 19, 2015

[25] M. Arnold. *Matching and Covering with Multiple-Output Patterns*. Tech. rep. 1-68340-44. Delft, The Netherlands: Delft University of Technology, 1999

[26] M. Arnold and H. Corporaal. "Automatic Detection of Recurring Operation Patterns". In: *Proceedings of the 7th International Workshop on Hardware/Software Codesign*. CODES'99. Rome, Italy: ACM, 1999, pp. 22–26. ISBN: 1-58113-132-1

[27] M. Arnold and H. Corporaal. "Designing Domain-Specific Processors". In: *Proceedings of the 9th International Symposium on Hardware/Software Codesign*. CODES'01. Copenhagen, Denmark: ACM, 2001, pp. 61–66. ISBN: 1-58113-364-2

[28] N. Arora, K. Chandramohan, N. Pothineni, and A. Kumar. "Instruction Selection in ASIP Synthesis Using Functional Matching". In: *Proceedings of the 23rd International Conference on VLSI Design*. VLSID'10. Washington, District of Colombia, USA: IEEE Computer Society, 2010, pp. 146–151

[29] M. A. Arslan and K. Kuchcinski. "Instruction Selection and Scheduling for DSP Kernels". In: *Microprocessors and Microsystems* 38.8, Part A (2014), pp. 803–813. ISSN: 0141-9331

[30] M. A. Arslan and K. Kuchcinski. "Instruction Selection and Scheduling for DSP Kernels on Custom Architectures". In: *Proceedings of the 16th EUROMICRO Conference on Digital System Design*. DSD'13. Santander, Cantabria, Spain: IEEE Computer Society, Sept. 4–6, 2013

[31] K. Atasu, G. Dündar, and C. özturan. "An Integer Linear Programming Approach for Identifying Instruction-Set Extensions". In: *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS'05. Jersey City, New Jersey, USA: ACM, 2005, pp. 172–177. ISBN: 1-59593-161-9

[32] K. Atasu, L. Pozzi, and P. Ienne. "Automatic Application-Specific Instruction-Set Extensions Under Microarchitectural Constraints". In: *Proceedings of the 40th Annual Design Automation Conference*. DAC'03. Anaheim, California, USA: ACM, 2003, pp. 256–261. ISBN: 1-58113-688-9

[33] M. Auslander and M. Hopkins. "An Overview of the PL.8 Compiler". In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN'82. Boston, Massachusetts, USA: ACM, 1982, pp. 22–31. ISBN: 0-89791-074-5

[34] M. W. Bailey and J. W. Davidson. "Automatic Detection and Diagnosis of Faults in Generated Code for Procedure Calls". In: *Transactions on Software Engineering* 29.11 (2003), pp. 1031–1042. ISSN: 0098-5589

[35] A. Balachandran, D. M. Dhamdhere, and S. Biswas. "Efficient Retargetable Code Generation Using Bottom-Up Tree Pattern Matching". In: *Computer Languages* 15.3 (1990), pp. 127–140. ISSN: 0096-0551

[36] M. Balakrishnan, P. C. P. Bhatt, and B. B. Madan. “An Efficient Retargetable Microcode Generator”. In: *Proceedings of the 19th Annual Workshop on Microprogramming*. MICRO 19. New York, New York, USA: ACM, 1986, pp. 44–53. ISBN: 0-8186-0736-X

[37] R. A. Ballance, A. B. MacCabe, and K. J. Ottenstein. “The Program Dependence Web: A Representation Supporting Control-, Data-, and Demand-Driven Interpretation of Imperative Languages”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. SIGPLAN’90. White Plains, New York, USA: ACM, 1990, pp. 257–271. ISBN: 0-89791-364-7

[38] S. Bansal and A. Aiken. “Automatic Generation of Peephole Superoptimizers”. In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS’06. San Jose, California, USA: ACM, 2006, pp. 394–403

[39] R. Barik, J. Zhao, and V. Sarkar. “Efficient Selection of Vector Instructions Using Dynamic Programming”. In: *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO. Washington, District of Colombia, USA: IEEE Computer Society, 2010, pp. 201–212

[40] S. Bashford and R. Leupers. “Constraint Driven Code Selection for Fixed-Point DSPs”. In: *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*. DAC’99. New Orleans, Louisiana, USA: ACM, 1999, pp. 817–822. ISBN: 1-58113-109-7

[41] L. Bauer, M. Shafique, and J. Henkel. “Run-Time Instruction Set Selection in a Transmutable Embedded Processor”. In: *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*. Washington, District of Colombia, USA: IEEE Computer Society, June 2008, pp. 56–61

[42] A. Bednarski and C. Kessler. “Energy-Optimal Integrated VLIW Code Generation”. In: *Proceedings of the 11th Workshop on Compilers for Parallel Computers*. CPC’04. Chiemsee, Bavaria, Germany, 2004, pp. 227–238

[43] A. Bednarski and C. W. Kessler. “Optimal Integrated VLIW Code Generation with Integer Linear Programming”. In: *Proceedings of the 12th International Euro-Par Conference*. Vol. 4128. Lecture Notes in Computer Science. Dresden, Germany: Springer, 2006, pp. 461–472

[44] M. O. Beg. “Combinatorial Problems in Compiler Optimization”. Doctoral thesis. Ontario, Canada: University of Waterloo, 2013

[45] N. Beldiceanu, M. Carlsson, and J. Rampon. *Global Constraint Catalog*. Feb. 24, 2014. URL: `http://www.emn.fr/z-info/sdemasse/gccat/` (visited on 2014-04-15)

[46] E. Bendersky. *A Deeper Look into the LLVM Code Generator: Part 1*. Feb. 25, 2013. URL: `http://eli.thegreenplace.net/2013/02/25/a-deeper-look-into-the-llvm-code-generator-part-1/` (visited on 2013-05-10)

[47] R. V. Bennett, A. C. Murray, B. Franke, and N. Topham. “Combining Source-to-Source Transformations and Processor Instruction Set Extensions for the Automated Design-Space Exploration of Embedded Systems”. In: *Proceedings of the SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES’07. San Diego, California, USA: ACM, 2007, pp. 83–92. ISBN: 978-1-59593-632-5

[48] I. Boehm. *HBURG*. 2007. URL: `http://www.bytelabs.org/hburg.html` (visited on 2014-02-11)

[49] J. Boender and C. Sacerdoti Coen. “On the Correctness of a Branch Displacement Algorithm”. In: *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’14)*. Ed. by E. Ábrahám and K. Havelund. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 605–619. ISBN: 978-3-642-54861-1

[50] B. Borchardt. “Code Selection by Tree Series Transducers”. In: *Proceedings of the 9th International Conference on Implementation and Application of Automata*. CIAA’04. Sophia Antipolis, France: Springer, 2004, pp. 57–67. ISBN: 978-3-540-24318-2

[51] A. Bougacha. *[LLVMdev] [RFC] Integer Saturation Intrinsics*. 2015-01-14. URL: `https://groups.google.com/forum/#!topic/llvm-dev/fHThmnh8zkI` (visited on 2015-06-09)

[52] D. Boulytchev. “BURS-Based Instruction Set Selection”. In: *Proceedings of the 6th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics*. PSI’06. Novosibirsk, Russia: Springer, 2007, pp. 431–437. ISBN: 978-3-540-70880-3

[53] D. Boulytchev and D. Lomov. “An Empirical Study of Retargetable Compilers”. In: *Proceedings of the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI’01)*. Ed. by D. Bjørner, M. Broy, and A. V. Zamulin. Vol. 2244. Lecture Notes in Computer Science. Springer, 2001, pp. 328–335. ISBN: 978-3-540-43075-9

[54] F. Brandner. “Completeness of Automatically Generated Instruction Selectors”. In: *Proceedings of the 21st International Conference on Application-Specific Systems, Architectures and Processors*. ASAP’10. Washington, District of Colombia, USA: IEEE Computer Society, 2010, pp. 175–182

[55] F. Brandner, D. Ebner, and A. Krall. “Compiler Generation from Structural Architecture Descriptions”. In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. CASES’07. Salzburg, Austria: ACM, 2007, pp. 13–22. ISBN: 978-1-59593-826-8

[56] M. Bravenboer and E. Visser. “Rewriting Strategies for Instruction Selection”. In: *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA’02)*. Ed. by S. Tison. Vol. 2378. Lecture Notes in Computer Science. Springer, 2002, pp. 237–251. ISBN: 978-3-540-43916-5

[57] P. Brisk, A. Kaplan, R. Kastner, and M. Sarrafzadeh. “Instruction Generation and Regularity Extraction for Reconfigurable Processors”. In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. CASES’02. Grenoble, France: ACM, 2002, pp. 262–269. ISBN: 1-58113-575-0

[58] P. Brisk, A. Nahapetian, and M. Sarrafzadeh. “Instruction Selection for Compilers That Target Architectures with Echo Instructions”. In: *Proceedings of the 8th International Workshop on Software and Compilers for Embedded Systems (SCOPES’04)*. Ed. by H. Schepers. Vol. 3199. Lecture Notes in Computer Science. Springer, 2004, pp. 229–243. ISBN: 978-3-540-23035-9

[59] P. Brown. “A Survey of Macro Processors”. In: *Annual Review in Automatic Programming* 6.2 (1969), pp. 37–88. ISSN: 0066-4138

[60] J. Bruno and R. Sethi. “Code Generation for a One-Register Machine”. In: *Journal of the ACM* 23.3 (1976), pp. 502–510. ISSN: 0004-5411

[61] S. Buchwald and A. Zwinkau. “Instruction Selection by Graph Transformation”. In: *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. CASES’10. Scottsdale, Arizona, USA: ACM, 2010, pp. 31–40. ISBN: 978-1-60558-903-9

[62] J. Cai, R. Paige, and R. Tarjan. “More Efficient Bottom-Up Multi-pattern Matching in Trees”. In: *Theoretical Computer Science* 106.1 (1992), pp. 21–60. ISSN: 0304-3975

[63] P. Canalda, L. Cognard, A. Despland, M. Jourdan, M. Mazaud, D. Parigot, F. Thomasset, and D. de Voluceau. *PAGODE: A Realistic Back-End Generator*. Tech. rep. Rocquencourt, France: INRIA, 1995

[64] Z. Cao, Y. Dong, and S. Wang. “Compiler Backend Generation for Application Specific Instruction Set Processors”. In: *Proceedings of the 9th Asian Symposium on Programming Languages and Systems (APLAS’11)*. Ed. by H. Yang. Vol. 7078. Lecture Notes in Computer Science. Springer, 2011, pp. 121–136. ISBN: 978-3-642-25317-1

[65] R. Castañeda Lozano, M. Carlsson, F. Drejhammar, and C. Schulte. “Constraint-Based Register Allocation and Instruction Scheduling”. In: *Proceedings of the 18th International Conference on the Principles and Practice of Constraint Programming (CP’12)*. Ed. by M. Milano. Vol. 7514. Lecture Notes in Computer Science. Springer, 2012, pp. 750–766. ISBN: 978-3-642-33557-0

[66] R. Castañeda Lozano, M. Carlsson, G. Hjort Blindell, and C. Schulte. “Combinatorial Spill Code Optimization and Ultimate Coalescing”. In: *Proceedings of the 14th SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*. LCTES’14. Edinburgh, United Kingdom, 2014, pp. 23–32

[67] R. G. Cattell. “Automatic Derivation of Code Generators from Machine Descriptions”. In: *Transactions on Programming Languages and Systems* 2.2 (1980), pp. 173–190. ISSN: 0164-0925

[68] R. G. G. Cattell. *A Survey and Critique of Some Models of Code Generation*. Tech. rep. Pittsburgh, Pennsylvania, USA: School of Computer Science, Carnegie Mellon University, 1979

[69] R. G. G. Cattell. “Formalization and Automatic Derivation of Code Generators”. AAI7815197. Doctoral thesis. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University, 1978

[70] R. G. Cattell, J. M. Newcomer, and B. W. Leverett. “Code Generation in a Machine-Independent Compiler”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’79. Denver, Colorado, USA: ACM, 1979, pp. 65–75. ISBN: 0-89791-002-8

[71] P. E. Ceruzzi. *A History of Modern Computing*. 2nd ed. Cambridge, Massachusetts, USA: MIT Press, 2003. ISBN: 978-0262532037

[72] J.-M. Chang and M. Pedram. “Register Allocation and Binding for Low Power”. In: *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference*. DAC’95. San Francisco, California, USA: ACM, 1995, pp. 29–35. ISBN: 0-89791-725-1

[73] D. R. Chase. “An Improvement to Bottom-Up Tree Pattern Matching”. In: *Proceedings of the 14th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’87. Munich, West Germany: ACM, 1987, pp. 168–177. ISBN: 0-89791-215-2

[74] T. Chen, F. Lai, and R. Shang. “A Simple Tree Pattern Matching Algorithm for Code Generator”. In: *Proceedings of the 19th Annual International Conference on Computer Software and Applications*. COMPSAC’95. Dallas, Texas, USA: IEEE Computer Society, 1995, pp. 162–167

[75] D. Cho, A. Ravi, G.-R. Uh, and Y. Paek. “Instruction Re-selection for Iterative Modulo Scheduling on High Performance Multi-issue DSPs”. In: *Emerging Directions in Embedded and Ubiquitous Computing*. Ed. by X. Zhou, O. Sokolsky, L. Yan, E.-S. Jung, Z. Shao, Y. Mu, D. Lee, D. Kim, Y.-S. Jeong, and C.-Z. Xu. Vol. 4097. Lecture Notes in Computer Science. Springer, 2006, pp. 741–754. ISBN: 978-3-540-36850-2

[76] T. W. Christopher, P. J. Hatcher, and R. C. Kukuk. “Using Dynamic Programming to Generate Optimized Code in a Graham-Glanville Style Code Generator”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’84. Montreal, Canada: ACM, 1984, pp. 25–36. ISBN: 0-89791-139-3

[77] N. Clark, A. Hormati, S. Mahlke, and S. Yehia. “Scalable Subgraph Mapping for Acyclic Computation Accelerators”. In: *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. CASES’06. Seoul, Korea: ACM, 2006, pp. 147–157. ISBN: 1-59593-543-6

[78] N. Clark, H. Zhong, and S. Mahlke. “Processor Acceleration Through Automated Instruction Set Customization”. In: *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 36. Washington, District of Colombia, USA: IEEE Computer Society, 2003, pp. 129–140. ISBN: 0-7695-2043-X

[79] C. Click and K. D. Cooper. “Combining Analyses, Combining Optimizations”. In: *Transactions on Programming Languages and Systems* 17.2 (Mar. 1995), pp. 181–196. ISSN: 0164-0925

[80] C. Click and M. Paleczny. “A Simple Graph-based Intermediate Representation”. In: *Papers from the SIGPLAN Workshop on Intermediate Representations*. IR’95. San Francisco, California, USA: ACM, 1995, pp. 35–49. ISBN: 0-89791-754-5

[81] R. Cole and R. Hariharan. “Tree Pattern Matching and Subset Matching in Randomized $O(n \log^3 m)$ Time”. In: *Proceedings of the 29th Annual Symposium on Theory of Computing*. STOC’97. El Paso, Texas, USA: ACM, 1997, pp. 66–75. ISBN: 0-89791-888-6

[82] J. Cong, Y. Fan, G. Han, and Z. Zhang. “Application-Specific Instruction Generation for Configurable Processor Architectures”. In: *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*. FPGA’04. Monterey, California, USA: ACM, 2004, pp. 183–189. ISBN: 1-58113-829-6

[83] M. E. Conway. “Proposal for an UNCOL”. In: *Communications of the ACM* 1.10 (1958), pp. 5–8. ISSN: 0001-0782

[84] S. A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *Proceedings of the 3rd Annual Symposium on Theory of Computing*. STOC’71. Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158

[85] K. D. Cooper and L. Torczon. *Engineering a Compiler*. 2nd ed. Burlington, Massachusetts, USA: Morgan Kaufmann, 2011. ISBN: 978-0120884780

[86] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. “An Improved Algorithm for Matching Large Graphs”. In: *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*. Springer, 2001, pp. 149–159

[87] R. Cordone, F. Ferrandi, D. Sciuto, and R. Wolfler Calvo. “An Efficient Heuristic Approach to Solve the Unate Covering Problem”. In: *Proceedings of the Conference and exhibition on Design, Automation and Test in Europe*. DATE’00. Washington, District of Colombia, USA: IEEE Computer Society, 2000, pp. 364–371

[88] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 3rd ed. Cambridge, Massachusetts, USA: MIT Press, 2009. ISBN: 978-0262033848

[89] *CoSy Compilers: Overview of Construction and Operation*. CoSy-8004-construct. ACE Associated Compiler Experts. 2003

[90] T. Crick, M. Brain, M. Vos, and J. Fitch. “Generating Optimal Code Using Answer Set Programming”. In: *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning*. LPNMR’09. Potsdam, Germany: Springer, 2009, pp. 554–559. ISBN: 978-3-642-04237-9

[91] J. W. Davidson and C. W. Fraser. “Code Selection Through Object Code Optimization”. In: *Transactions on Programming Languages and Systems* 6.4 (1984), pp. 505–526. ISSN: 0164-0925

[92] J. W. Davidson and C. W. Fraser. “Eliminating Redundant Object Code”. In: *Proceedings of the 9th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’82. Albuquerque, New Mexico, USA: ACM, 1982, pp. 128–132. ISBN: 0-89791-065-6

[93] J. W. Davidson and C. W. Fraser. “The Design and Application of a Retargetable Peephole Optimizer”. In: *Transactions on Programming Languages and Systems* 2.2 (1980), pp. 191–202. ISSN: 0164-0925

[94] A. Despland, M. Mazaud, and R. Rakotozafy. “Code Generator Generation Based on Template-Driven Target Term Rewriting”. In: *Rewriting Techniques and Applications*. Bordeaux, France: Springer, 1987, pp. 105–120. ISBN: 0-387-17220-3

[95] A. Despland, M. Mazaud, and R. Rakotozafy. “Using Rewriting Techniques to Produce Code Generators and Proving Them Correct”. In: *Science of Computer Programming* 15.1 (1990), pp. 15–54. ISSN: 0167-6423

[96] L. P. Deutsch and A. M. Schiffman. “Efficient Implementation of the Smalltalk-80 System”. In: *Proceedings of the 11th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’84. Salt Lake City, Utah, USA: ACM, 1984, pp. 297–302. ISBN: 0-89791-125-3

[97] J. Dias and N. Ramsey. “Automatically Generating Instruction Selectors Using Declarative Machine Descriptions”. In: *Proceedings of the 37th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’10. Madrid, Spain: ACM, 2010, pp. 403–416. ISBN: 978-1-60558-479-9

[98] J. Dias and N. Ramsey. “Converting Intermediate Code to Assembly Code Using Declarative Machine Descriptions”. In: *Proceedings of the 15th International Conference on Compiler Construction*. CC’06. Vienna, Austria: Springer, 2006, pp. 217–231. ISBN: 3-540-33050-X, 978-3-540-33050-9

[99] A. Dold, T. Gaul, V. Vialard, and W. Zimmermann. “ASM-Based Mechanized Verification of Compiler Back-Ends”. In: *Proceedings of the 5th International Workshop on Abstract State Machines*. Ed. by U. Glässer and P. H. Schmitt. Magdeburg, Germany, 1998, pp. 50–67

[100] M. K. Donegan, R. E. Noonan, and S. Feyock. “A Code Generator Generator Language”. In: *Proceedings of the 1979 SIGPLAN Symposium on Compiler Construction*. SIGPLAN’79. Denver, Colorado, USA: ACM, 1979, pp. 58–64

[101] M. K. Donegan. “An Approach to the Automatic Generation of Code Generators”. AAI7321548. Doctoral thesis. Houston, Texas, USA: Rice University, 1973

[102] M. Dorigo and T. Stützle. “Ant Colony Optimization: Overview and Recent Advances”. In: *Handbook of Metaheuristics*. Ed. by M. Gendreau and J.-Y. Potvin. 2nd ed. Vol. 146. International Series in Operations Research & Management Science. Springer, 2010. Chap. 8, pp. 227–263. ISBN: 978-1-4419-116-1

[103] M. Dubiner, Z. Galil, and E. Magen. “Faster Tree Pattern Matching”. In: *Journal of the ACM* 41.2 (1994), pp. 205–213. ISSN: 0004-5411

[104] J. Earley. “An Efficient Context-Free Parsing Algorithm”. In: *Communications of the ACM* 13.2 (1970), pp. 94–102. ISSN: 0001-0782

[105] D. Ebner, F. Brandner, B. Scholz, A. Krall, P. Wiedermann, and A. Kadlec. "Generalized Instruction Selection Using SSA-Graphs". In: *Proceedings of the SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES'08. Tucson, Arizona, USA: ACM, 2008, pp. 31–40. ISBN: 978-1-60558-104-0

[106] E. Eckstein, O. König, and B. Scholz. "Code Instruction Selection Based on SSA-Graphs". In: *Proceedings of the 7th International Workshop on Software and Compilers for Embedded Systems (SCOPES'03)*. Ed. by A. Krall. Vol. 2826. Lecture Notes in Computer Science. Springer, 2003, pp. 49–65. ISBN: 978-3-540-20145-8

[107] T. J. Edler von Koch, I. Böhm, and B. Franke. "Integrated Instruction Selection and Register Allocation for Compact Code Generation Exploiting Freeform Mixing of 16- and 32-bit Instructions". In: *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO'10. Toronto, Ontario, Canada: ACM, 2010, pp. 180–189. ISBN: 978-1-60558-635-9

[108] M. Elson and S. T. Rake. "Code-Generation Technique for Large-Language Compilers". In: *IBM Systems Journal* 9.3 (1970), pp. 166–188. ISSN: 0018-8670

[109] H. Emmelmann, F.-W. Schröer, and R. Landwehr. "BEG: A Generator for Efficient Back Ends". In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. SIGPLAN'89. Portland, Oregon, USA: ACM, 1989, pp. 227–237. ISBN: 0-89791-306-X

[110] H. Emmelmann. "Code Selection by Regularly Controlled Term Rewriting". In: *Code Generation—Concepts, Tools, Techniques*. Ed. by R. Giegerich and S. L. Graham. Springer, 1992, pp. 3–29. ISBN: 978-3-540-19757-7

[111] H. Emmelmann. "Testing Completeness of Code Selector Specifications". In: *Proceedings of the 4th International Conference on Compiler Construction*. CC'92. Springer, 1992, pp. 163–175. ISBN: 3-540-55984-1

[112] J. Engelfriet, Z. Fülöp, and H. Vogler. "Bottom-Up and Top-Down Tree Series Transformations". In: *Journal of Automata, Languages and Combinatorics* 7.1 (July 2001), pp. 11–70. ISSN: 1430-189X

[113] D. R. Engler. "VCODE: A Retargetable, Extensible, Very Fast Dynamic Code Generation System". In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. SIGPLAN'96. Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 160–170. ISBN: 0-89791-795-2

[114] D. R. Engler and T. A. Proebsting. "DCG: An Efficient, Retargetable Dynamic Code Generation System". In: *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS VI. San Jose, California, USA: ACM, 1994, pp. 263–272. ISBN: 0-89791-660-3

[115] M. V. Eriksson, O. Skoog, and C. W. Kessler. "Optimal vs. Heuristic Integrated Code Generation for Clustered VLIW Architectures". In: *Proceedings of the 11th International Workshop on Software & Compilers for Embedded Systems*. SCOPES'08. Munich, Germany: ACM, 2008, pp. 11–20

[116] M. Eriksson and C. Kessler. “Integrated Code Generation for Loops”. In: *Transactions on Embedded Computing Systems* 11S.1 (June 2012), 19:1–19:24. ISSN: 1539-9087

[117] M. A. Ertl. “Optimal Code Selection in DAGs”. In: *Proceedings of the 26th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’99. San Antonio, Texas, USA: ACM, 1999, pp. 242–249. ISBN: 1-58113-095-3

[118] M. A. Ertl, K. Casey, and D. Gregg. “Fast and Flexible Instruction Selection with On-Demand Tree-Parsing Automata”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’06. Ottawa, Ontario, Canada: ACM, 2006, pp. 52–60. ISBN: 1-59593-320-4

[119] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu. “Incremental Graph Pattern Matching”. In: *Proceedings of the SIGMOD International Conference on Management of Data*. SIGMOD’11. Athens, Greece: ACM, 2011, pp. 925–936. ISBN: 978-1-4503-0661-4

[120] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. “Graph Pattern Matching: From Intractable to Polynomial Time”. In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 264–275. ISSN: 2150-8097

[121] S. Farfeleder, A. Krall, E. Steiner, and F. Brandner. “Effective Compiler Generation by Architecture Description”. In: *Proceedings of the SIGPLAN/SIGBED Conference on Language, Compilers, and Tool support for Embedded Systems*. LCTES’06. Ottawa, Ontario, Canada: ACM, 2006, pp. 145–152. ISBN: 1-59593-362-X

[122] R. Farrow. “Experience with an Attribute Grammar-based Compiler”. In: *Proceedings of the 9th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’82. Albuquerque, New Mexico, USA: ACM, 1982, pp. 95–107. ISBN: 0-89791-065-6

[123] A. Fauth, M. Freericks, and A. Knoll. “Generation of Hardware Machine Models from Instruction Set Descriptions”. In: *Proceedings of the Workshop on VLSI Signal Processing*. VI’93. Washington, District of Colombia, USA: IEEE Computer Society, 1993, pp. 242–250

[124] A. Fauth, J. Van Praet, and M. Freericks. “Describing Instruction Set Processors Using nML”. In: *Proceedings of the European Conference on Design and Test*. EDTC’95. Washington, District of Colombia, USA: IEEE Computer Society, 1995, pp. 503–507. ISBN: 0-8186-7039-8

[125] A. Fauth, G. Hommel, A. Knoll, and C. Müller. “Global Code Selection of Directed Acyclic Graphs”. In: *Proceedings of the 5th International Conference on Compiler Construction*. CC’94. Springer, 1994, pp. 128–142. ISBN: 3-540-57877-3

[126] J. Feldman and D. Gries. “Translator Writing Systems”. In: *Communications of the ACM* 11.2 (1968), pp. 77–113. ISSN: 0001-0782

[127] C. Ferdinand, H. Seidl, and R. Wilhelm. “Tree Automata for Code Selection”. In: *Acta Informatica* 31.9 (1994), pp. 741–760. ISSN: 0001-5903

[128] M. Fernández and N. Ramsey. “Automatic Checking of Instruction Specifications”. In: *Proceedings of the 19th International Conference on Software Engineering*. ICSE’97. Boston, Massachusetts, USA: ACM, 1997, pp. 326–336. ISBN: 0-89791-914-9

[129] J. Ferrante, K. J. Ottenstein, and J. D. Warren. “The Program Dependence Graph and Its Use in Optimization”. In: *Transactions on Programming Languages and Systems* 9.3 (July 1987), pp. 319–349. ISSN: 0164-0925

[130] C. N. Fischer, R. K. Cytron, and R. J. J. LeBlanc. *Crafting a Compiler*. London, England: Pearson, 2009. ISBN: 978-0138017859

[131] A. Floch, C. Wolinski, and K. Kuchcinski. “Combined Scheduling and Instruction Selection for Processors with Reconfigurable Cell Fabric”. In: *Proceedings of the 21st International Conference on Application-Specific Systems, Architectures and Processors*. ASAP’10. Washington, District of Colombia, USA: IEEE Computer Society, 2010, pp. 167–174

[132] R. W. Floyd. “Algorithm 97: Shortest Path”. In: *Communications of the ACM* 5.6 (1962), p. 345. ISSN: 0001-0782

[133] R. W. Floyd. “An Algorithm for Coding Efficient Arithmetic Operations”. In: *Communications of the ACM* 4.1 (1961), pp. 42–51. ISSN: 0001-0782

[134] C. W. Fraser. “A Language for Writing Code Generators”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’89. Portland, Oregon, USA: ACM, 1989, pp. 238–245. ISBN: 0-89791-306-X

[135] C. W. Fraser and A. L. Wendt. “Automatic Generation of Fast Optimizing Code Generators”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’88. Atlanta, Georgia, USA: ACM, 1988, pp. 79–84. ISBN: 0-89791-269-1

[136] C. W. Fraser. “A Compact, Machine-Independent Peephole Optimizer”. In: *Proceedings of the 6th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’79. San Antonio, Texas, USA: ACM, 1979, pp. 1–6

[137] C. W. Fraser, D. R. Hanson, and T. A. Proebsting. “Engineering a Simple, Efficient Code-Generator Generator”. In: *Letters on Programming Languages and Systems* 1.3 (1992), pp. 213–226. ISSN: 1057-4514

[138] C. W. Fraser, R. R. Henry, and T. A. Proebsting. “BURG—Fast Optimal Instruction Selection and Tree Parsing”. In: *SIGPLAN Notices* 27.4 (1992), pp. 68–76. ISSN: 0362-1340

[139] C. W. Fraser and T. A. Proebsting. “Finite-State Code Generation”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’99. Atlanta, Georgia, USA: ACM, 1999, pp. 270–280. ISBN: 1-58113-094-5

[140] C. W. Fraser. “A Knowledge-Based Code Generator Generator”. In: *Proceedings of the Symposium on Artificial Intelligence and Programming Languages*. New York, New York, USA: ACM, 1977, pp. 126–129

[141] C. W. Fraser. “Automatic Generation of Code Generators”. Doctoral thesis. New Haven, Connecticut, USA: Yale University, 1977

[142] S. Fröhlich, M. Gotschlich, U. Krebelder, and B. Wess. “Dynamic Trellis Diagrams for Optimized DSP Code Generation”. In: *Proceedings of the International Symposium on Circuits and Systems*. ISCAS’99. Washington, District of Colombia, USA: IEEE Computer Society, 1999, pp. 492–495

[143] B. Gallagher. *The State of the Art in Graph-Based Pattern Matching*. Tech. rep. UCRL-TR-220300. Livermore, California, USA: Lawrence Livermore National Laboratory, Mar. 31, 2006

[144] C. Galuzzi and K. Bertels. “The Instruction-Set Extension Problem: A Survey”. In: *Transactions on Reconfigurable Technology and Systems* 4.2 (May 2011), 18:1–18:28. ISSN: 1936-7406

[145] M. Ganapathi. “Prolog Based Retargetable Code Generation”. In: *Computer Languages* 14.3 (1989), pp. 193–204. ISSN: 0096-0551

[146] M. Ganapathi. “Retargetable Code Generation and Optimization Using Attribute Grammars”. AAI8107834. Doctoral thesis. Madison, Wisconsin, USA: The University of Wisconsin–Madison, 1980

[147] M. Ganapathi and C. N. Fischer. “Affix Grammar Driven Code Generation”. In: *Transactions on Programming Languages and Systems* 7.4 (1985), pp. 560–599. ISSN: 0164-0925

[148] M. Ganapathi and C. N. Fischer. “Description-Driven Code Generation Using Attribute Grammars”. In: *Proceedings of the 9th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’82. Albuquerque, New Mexico, USA: ACM, 1982, pp. 108–119. ISBN: 0-89791-065-6

[149] M. Ganapathi and C. N. Fischer. *Instruction Selection by Attributed Parsing*. Tech. rep. No. 84-256. Stanford, California, USA: Stanford University, 1984

[150] M. Ganapathi, C. N. Fischer, and J. L. Hennessy. “Retargetable Compiler Code Generation”. In: *Computing Surveys* 14.4 (1982), pp. 573–592. ISSN: 0360-0300

[151] M. R. Garey and D. S. Johnson. *Computers and Intractability*. New York, New York, USA: W. H. Freeman and Company, 1979

[152] C. H. Gebotys. “An Efficient Model for DSP Code Generation: Performance, Code Size, Estimated Energy”. In: *Proceedings of the 10th International Symposium on System Synthesis*. ISSS’97. Antwerp, Belgium: IEEE Computer Society, 1997, pp. 41–47. ISBN: 0-8186-7949-2

[153] C. H. Gebotys. “Low Energy Memory and Register Allocation Using Network Flow”. In: *Proceedings of the 34th Annual Design Automation Conference*. DAC’97. Anaheim, California, USA: ACM, 1997, pp. 435–440. ISBN: 0-89791-920-3

[154] F. Gecseg and M. Steinby. *Tree Automata*. Budapest, Hungary: Akadémiai Kiadó, 1984. ISBN: 978-96305317-02

[155] D. Genin, J. De Moortel, D. Desmet, and E. Van de Velde. “System Design, Optimization and Intelligent Code Generation for Standard Digital Signal Processors”. In: *Proceedings of the International Symposium on Circuits and Systems*. ISCAS’90. Washington, District of Colombia, USA: IEEE Computer Society, 1989, pp. 565–569

[156] R. Giegerich. “A Formal Framework for the Derivation of Machine-Specific Optimizers”. In: *Transactions on Programming Languages and Systems* 5.3 (1983), pp. 478–498. ISSN: 0164-0925

[157] R. Giegerich and K. Schmal. “Code Selection Techniques: Pattern Matching, Tree Parsing, and Inversion of Derivors”. In: *Proceedings of the 2nd European Symposium on Programming*. Ed. by H. Ganzinger. Vol. 300. ESOP’88. Nancy, France: Springer, 1998, pp. 247–268. ISBN: 978-3-540-19027-1

[158] R. S. Glanville and S. L. Graham. “A New Method for Compiler Code Generation”. In: *Proceedings of the 5th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’78. Tucson, Arizona, USA: Springer, 1978, pp. 231–254

[159] E. I. Goldberg, L. P. Carloni, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. “Negative Thinking in Branch-and-Bound: The Case of Unate Covering”. In: *Transactions of Computer-Aided Design of Integrated Ciruits and Systems* 19.3 (2006), pp. 281–294. ISSN: 0278-0070

[160] K. J. Gough. *Bottom-Up Tree Rewriting Tool MBURG*. Tech. rep. Brisbane, Australia: Faculty of Information Technology, Queensland University of Technology, July 18, 1995

[161] K. J. Gough and J. Ledermann. “Optimal Code-Selection using MBURG”. In: *Proceedings of the 20th Australasian Computer Science Conference*. ACSC’97. Sydney, Australia, 1997

[162] K. Gough. “Reconceptualizing Bottom-Up Tree Rewriting”. In: *Patterns, Programming and Everything*. Ed. by K. K. Breitman and R. N. Horspool. Springer, 2012, pp. 31–44. ISBN: 978-1-4471-2349-1

[163] S. L. Graham. “Table-Driven Code Generation”. In: *Computer* 13.8 (1980), pp. 25–34. ISSN: 0018-9162

[164] S. L. Graham, R. R. Henry, and R. A. Schulman. “An Experiment in Table Driven Code Generation”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’82. Boston, Massachusetts, USA: ACM, 1982, pp. 32–43. ISBN: 0-89791-074-5

[165] T. Granlund and R. Kenner. “Eliminating Branches Using a Superoptimizer and the GNU C Compiler”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’92. San Francisco, California, USA: ACM, 1992, pp. 341–352. ISBN: 0-89791-4759

[166] Y. Guo, G. J. Smit, H. Broersma, and P. M. Heysters. “A Graph Covering Algorithm for a Coarse Grain Reconfigurable System”. In: *Proceedings of the SIGPLAN Conference on Language, Compiler, and Tools for Embedded Systems*. LCTES’03. San Diego, California, USA: ACM, 2003, pp. 199–208. ISBN: 1-58113-647-1

[167] S. Z. Hanono. “AVIV: A Retargetable Code Generator for Embedded Processors”. AAI7815197. Doctoral thesis. Cambridge, Massachusetts, USA: Massachusetts Institute of Technology, 1999

[168] S. Hanono and S. Devadas. “Instruction Selection, Resource Allocation, and Scheduling in the AVIV Retargetable Code Generator”. In: *Proceedings of the 35th Annual Design Automation Conference*. DAC’98. San Francisco, California, USA: ACM, 1998, pp. 510–515. ISBN: 0-89791-964-5

[169] D. R. Hanson and C. W. Fraser. *A Retargetable C Compiler: Design and Implementation*. Boston, Massachusetts, USA: Addison-Wesley, 1995. ISBN: 978-0805316704

[170] W. H. Harrison. “A New Strategy for Code Generation the General-Purpose Optimizing Compiler”. In: *Transactions Software Engineering* 5.4 (1979), pp. 367–373. ISSN: 0098-5589

[171] T. Harwood, K. Kumar, and N. Bereton. *JBURG*. 2013. URL: `http://jburg.sourceforge.net/` (visited on 2014-02-11)

[172] P. J. Hatcher and T. W. Christopher. “High-Quality Code Generation via Bottom-Up Tree Pattern Matching”. In: *Proceedings of the 13th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’86. St. Petersburg Beach, Florida, USA: ACM, 1986, pp. 119–130

[173] P. Hatcher. “The Equational Specification of Efficient Compiler Code Generation”. In: *Computer Languages* 16.1 (1991), pp. 81–95. ISSN: 0096-0551

[174] P. Hatcher and J. W. Tuller. “Efficient Retargetable Compiler Code Generation”. In: *Proceedings for the International Conference on Computer Languages*. Miami Beach, Florida, USA: IEEE Computer Society, 1988, pp. 25–30. ISBN: 0-8186-0874-9

[175] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. 5th ed. Burlington, Massachusetts, USA: Morgan Kaufmann, 2011

[176] R. R. Henry. *Encoding Optimal Pattern Selection in Table-Driven Bottom-Up Tree-Pattern Matcher*. Tech. rep. 89-02-04. Seattle, Washington, USA: University of Washington, 1989

[177] R. R. Henry. “Graham-Glanville Code Generators”. UCB/CSD-84-184. Doctoral thesis. Berkeley, California, USA: EECS Department, University of California, May 1984

[178] T. Hino, Y. Suzuki, T. Uchida, and Y. Itokawa. “Polynomial Time Pattern Matching Algorithm for Ordered Graph Patterns”. In: *Proceedings of the 22nd International Conference on Inductive Logic Programming*. ILP’12. Dubrovnik, Croatia: Springer, 2012, pp. 86–101

[179] C. M. Hoffmann and M. J. O’Donnell. “Pattern Matching in Trees”. In: *Journal of the ACM* 29.1 (1982), pp. 68–95. ISSN: 0004-5411

[180] M. Hohenauer, C. Schumacher, R. Leupers, G. Ascheid, H. Meyr, and H. van Someren. “Retargetable Code Optimization with SIMD Instructions”. In: *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS’06. Seoul, Korea: ACM, 2006, pp. 148–153. ISBN: 1-59593-370-0

[181] J. N. Hooker. “Resolution vs. Cutting Plane Solution of Inference Problems: Some Computational Experience”. In: *Operations Research Letters* 7.1 (Feb. 1988), pp. 1–7. ISSN: 0167-6377

[182] R. Hoover and K. Zadeck. “Generating Machine Specific Optimizing Compilers”. In: *Proceedings of the 23rd SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’96. St. Petersburg Beach, Florida, USA: ACM, 1996, pp. 219–229. ISBN: 0-89791-769-3

[183] A. Hormati, N. Clark, and S. Mahlke. “Exploiting Narrow Accelerators with Data-Centric Subgraph Mapping”. In: *Proceedings of the International Symposium on Code Generation and Optimization*. CGI’07. Washington, District of Colombia, USA: IEEE Computer Society, 2007, pp. 341–353

[184] R. N. Horspool. “An Alternative to the Graham-Glanville Code-Generation Method”. In: *Software* 4.3 (May 1987), pp. 33–39. ISSN: 0740-7459

[185] I. Huang and A. M. Despain. “Synthesis of Application Specific Instruction Sets”. In: *Transactions on Computer Aided Design of Integrated Circuits and Systems* 14.6 (June 1995), pp. 663–675

[186] *Intel 64 and IA-32 Architectures: Software Developer’s Manual*. Intel. Apr. 2015

[187] J. Janoušek and J. Málek. “Target Code Selection by Tilling AST with the Use of Tree Pattern Pushdown Automaton”. In: *Proceedings of the 3rd Symposium on Languages, Applications and Technologies*. Ed. by M. J. V. Pereira, J. P. Leal, and A. Simões. Vol. 38. SLATE’14. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 159–165. ISBN: 978-3-939897-68-2

[188] X. Jiang and H. Bunke. “On the Coding of Ordered Graphs”. In: *Computing* 61.1 (1998), pp. 23–38. ISSN: 0010-485X

[189] X. Jiang and H. Bunke. “Marked Subgraph Isomorphism of Ordered Graphs”. In: *Advances in Pattern Recognition*. Ed. by A. Amin, D. Dori, P. Pudil, and H. Freeman. Vol. 1451. Lecture Notes in Computer Science. Springer, 1998, pp. 122–131. ISBN: 978-3-540-64858-1

[190] X. Jiang and H. Bunke. “Including Geometry in Graph Representations: A Quadratic-Time Graph Isomorphism Algorithm and Its Applications”. In: *Advances in Structural and Syntactical Pattern Recognition*. Ed. by P. Perner, P. Wang, and A. Rosenfeld. Vol. 1121. Lecture Notes in Computer Science. Springer, 1996, pp. 110–119. ISBN: 978-3-540-61577-4

[191] S. C. Johnson. “A Portable Compiler: Theory and Practice”. In: *Proceedings of the 5th SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL’78. Tucson, Arizona, USA: ACM, 1978, pp. 97–104

[192] S. C. Johnson. “A Tour Through the Portable C Compiler”. In: *Unix Programmer’s Manual*. 7th ed. Vol. 2B. Murray Hill, New Jersey, USA: AT&T Bell Laboratories, 1981. Chap. 33

[193] R. Joshi, G. Nelson, and K. Randall. “Denali: A Goal-Directed Superoptimizer”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’02. Berlin, Germany: ACM, 2002, pp. 304–314. ISBN: 1-58113-463-0

[194] R. Joshi, G. Nelson, and Y. Zhou. “Denali: A Practical Algorithm for Generating Optimal Code”. In: *Transactions on Programming Languages and Systems* 28.6 (2006), pp. 967–989. ISSN: 0164-0925

[195] K. Kang. “A Study on Generating an Efficient Bottom-Up Tree Rewrite Machine for JBURG”. In: *Proceedings of the International Conference on Computational Science and Its Applications (ICCSA’04)*. Ed. by A. Laganá, M. Gavrilova, V. Kumar, Y. Mun, C. Tan, and O. Gervasi. Vol. 3043. Lecture Notes in Computer Science. Assisi, Italy: Springer, 2004, pp. 65–72. ISBN: 978-3-540-22054-1

[196] K. Kang and K. Choe. *On the Automatic Generation of Instruction Selector Using Bottom-Up Tree Pattern Matching*. Tech. rep. CS/TR-95-93. Daejeon, South Korea: Korea Advanced Institute of Science and Technology, 1995

[197] R. M. Karp, R. E. Miller, and A. L. Rosenberg. “Rapid Identification of Repeated Patterns in Strings, Trees and Arrays”. In: *Proceedings of the 4th Annual Symposium on Theory of Computing*. STOC’72. Denver, Colorado, USA: ACM, 1972, pp. 125–136

[198] R. Kastner, A. Kaplan, S. O. Memik, and E. Bozorgzadeh. “Instruction Generation for Hybrid Reconfigurable Systems”. In: *Transactions on Design Automation of Electronic Systems* 7.4 (2002), pp. 605–627. ISSN: 1084-4309

[199] C. W. Kessler and A. Bednarski. “A Dynamic Programming Approach to Optimal Integrated Code Generation”. In: *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES’01. New York, New York, USA: ACM, 2001, pp. 165–174

[200] C. W. Kessler and A. Bednarski. “Optimal Integrated Code Generation for Clustered VLIW Architectures”. In: *Proceedings of the SIGPLAN Joint Conference on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems*. LCTES/SCOPES’02. New York, New York, USA: ACM, 2002, pp. 102–111

[201] P. B. Kessler. “Discovering Machine-Specific Code Improvements”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’86. Palo Alto, California, USA: ACM, 1986, pp. 249–254. ISBN: 0-89791-197-0

[202] R. R. Kessler. “PEEP: An Architectural Description Driven Peephole Optimizer”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’84. Montreal, Canada: ACM, 1984, pp. 106–110. ISBN: 0-89791-139-3

[203] K. Keutzer. “DAGON: Technology Binding and Local Optimization by DAG Matching”. In: *Proceedings of the 24th ACM/IEEE Design Automation Conference*. DAC’87. New York, New York, USA: ACM, 1987, pp. 341–347

[204] R. El-Khalil and A. D. Keromytis. “Hydan: Hiding Information in Program Binaries”. In: *Proceedings of the 6th International Conference on Information and Communications Security (ICICS’04)*. Ed. by J. Lopez, S. Qing, and E. Okamoto. Vol. 3269. Lecture Notes in Computer Science. Springer, 2004, pp. 187–199. ISBN: 978-3-540-23563-7

[205] U. Khedker. “Workshop on Essential Abstractions in GCC”. Lecture. GCC Resource Center, Department of Computer Science and Engineering, IIT Bombay. Bombay, India, June 30–July 3, 2012

[206] S. Kim and H. Han. “Efficient SIMD Code Generation for Irregular Kernels”. In: *Proceedings of the 17th SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP’12. New Orleans, Louisiana, USA: ACM, 2012, pp. 55–64. ISBN: 978-1-4503-1160-1

[207] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220.4598 (1983), pp. 671–680

[208] D. E. Knuth. “On the Translation of Languages From Left to Right”. In: *Information and Control* 8 (6 Dec. 1965), pp. 607–639

[209] D. E. Knuth. “Semantics of Context-Free Languages”. In: *Mathematical Systems Theory* 2.2 (1968), pp. 127–145

[210] D. E. Knuth, J. H. J. Morris, and V. R. Pratt. “Fast Pattern Matching in Strings”. In: *SIAM Journal of Computing* 6.2 (1977), pp. 323–350. ISSN: 0097-5397

[211] D. Koes. “Towards a More Principled Compiler: Register Allocation and Instruction Selection Revisited”. Doctoral thesis. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University, 2009

[212] D. R. Koes and S. C. Goldstein. “Near-Optimal Instruction Selection on DAGs”. In: *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO’08. Boston, Massachusetts, USA: ACM, 2008, pp. 45–54. ISBN: 978-1-59593-978-4

[213] M. Kong, R. Veras, K. Stock, F. Franchetti, L.-N. Pouchet, and P. Sadayappan. “When Polyhedral Transformations Meet SIMD Code Generation”. In: *Proceedings of the 34th SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’13. Seattle, Washington, USA: ACM, 2013, pp. 127–138. ISBN: 978-1-4503-2014-6

[214] A. Krall and S. Lelait. “Compilation Techniques for Multimedia Processors”. In: *International Journal of Parallel Programming* 28.4 (2000), pp. 347–361. ISSN: 0885-7458

[215] W. Kreuzer, W. Gotschlich, and B. Wess. “REDACO: A Retargetable Data Flow Graph Compiler for Digital Signal Processors”. In: *Proceedings of the International Conference on Signal Processing Applications and Technology*. ICSPAT’96. Alameda, California, USA: Miller Freeman, 1996, pp. 742–746

[216] A. Krishnaswamy and R. Gupta. “Profile Guided Selection of ARM and Thumb Instructions”. In: *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems*. LCTES/SCOPES’02. Berlin, Germany: ACM, 2002, pp. 56–64. ISBN: 1-58113-527-0

[217] E. B. Krissinel and K. Henrick. “Common Subgraph Isomorphism Detection by Backtracking Search”. In: *Software–Practice & Experience* 34.6 (2004), pp. 591–607. ISSN: 0038-0644

[218] D. W. Krumme and D. H. Ackley. “A Practical Method for Code Generation Based on Exhaustive Search”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’82. Boston, Massachusetts, USA: ACM, 1982, pp. 185–196. ISBN: 0-89791-074-5

[219] P. Kulkarni, W. Zhao, S. Hines, D. Whalley, X. Yuan, R. v. Engelen, K. Gallivan, J. Hiser, J. Davidson, B. Cai, M. Bailey, H. Moon, K. Cho, and Y. Paek. "VISTA: VPO Interactive System for Tuning Applications". In: *Transactions on Embedded Computer Systems* 5.4 (Nov. 2006), pp. 819–863. ISSN: 1539-9087

[220] R. Landwehr, H.-S. Jansohn, and G. Goos. "Experience with an Automatic Code Generator Generator". In: *Proceedings of the 1982 SIGPLAN Symposium on Compiler Construction*. SIGPLAN'82. Boston, Massachusetts, USA: ACM, 1982, pp. 56–66. ISBN: 0-89791-074-5

[221] M. Langevin and E. Cerny. "An Automata-Theoretic Approach to Local Microcode Generation". In: *Proceedings of the 4th European Conference on Design Automation with the European Event in ASIC Design*. EDAC'93. Washington, District of Colombia, USA: IEEE Computer Society, 1993, pp. 94–98

[222] D. Lanneer, F. Catthoor, G. Goossens, M. Pauwels, J. Van Meerbergen, and H. De Man. "Open-Ended System for High-Level Synthesis of Flexible Signal Processors". In: *Proceedings of the Conference on European Design Automation*. EURO-DAC'90. Glasgow, Scotland: IEEE Computer Society, 1990, pp. 272–276. ISBN: 0-8186-2024-2

[223] S. Larsen and S. Amarasinghe. "Exploiting Superword Level Parallelism with Multimedia Instruction Sets". In: *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI'00. Vancouver, British Columbia, Canada: ACM, 2000, pp. 145–156. ISBN: 1-58113-199-2

[224] C. Lattner and V. Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*. CGO'04. Palo Alto, California, USA: IEEE Computer Society, 2004, pp. 75–86. ISBN: 0-7695-2102-9

[225] C. Lee, J. K. Lee, T. Hwang, and S. Tsai. "Compiler Optimization on VLIW Instruction Scheduling for Low Power". In: *Transactions on Design Automation of Electronic Systems* 8.2 (Apr. 2003), pp. 252–268. ISSN: 1084-4309

[226] R. Leupers and P. Marwedel. "Instruction Selection for Embedded DSPs with Complex Instructions". In: *Proceedings of the Conference on European Design Automation*. EURO-DAC/EURO-VHDL'96. Geneva, Switzerland: IEEE Computer Society, 1996, pp. 200–205. ISBN: 0-8186-7573-X

[227] R. Leupers. "Code Generation for Embedded Processors". In: *Proceedings of the 13th International Symposium on System Synthesis*. ISSS'00. Madrid, Spain: IEEE Computer Society, 2000, pp. 173–178. ISBN: 1-58113-267-0

[228] R. Leupers. "Code Selection for Media Processors with SIMD Instructions". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE'00. Paris, France: ACM, 2000, pp. 4–8. ISBN: 1-58113-244-1

[229] R. Leupers and S. Bashford. "Graph-Based Code Selection Techniques for Embedded Processors". In: *Transactions on Design Automation of Electronic Systems* 5 (4 2000), pp. 794–814. ISSN: 1084-4309

[230] R. Leupers and P. Marwedel. “Retargetable Code Generation Based on Structural Processor Description”. In: *Design Automation for Embedded Systems* 3.1 (1998), pp. 75–108. ISSN: 0929-5585

[231] R. Leupers and P. Marwedel. *Retargetable Compiler Technology for Embedded Systems*. Dordrecht, Netherlands: Kluwer Academic Publishers, 2001. ISBN: 0-7923-7578-5

[232] R. Leupers and P. Marwedel. “Retargetable Generation of Code Selectors from HDL Processor Models”. In: *Proceedings of the European Design and Test Conference*. EDTC’97. Paris, France: IEEE Computer Society, 1997, pp. 140–144

[233] R. Leupers and P. Marwedel. “Time-Constrained Code Compaction for DSPs”. In: *Proceedings of the 8th International Symposium on System Synthesis*. ISSS’95. Cannes, France: ACM, 1995, pp. 54–59. ISBN: 0-89791-771-5

[234] B. W. Leverett, R. G. G. Cattell, S. O. Hobbs, J. M. Newcomer, A. H. Reiner, B. R. Schatz, and W. A. Wulf. “An Overview of the Production-Quality Compiler-Compiler Project”. In: *Computer* 13.8 (1980), pp. 38–49. ISSN: 0018-9162

[235] S. Liao, K. Keutzer, S. Tjiang, and S. Devadas. “A New Viewpoint on Code Generation for Directed Acyclic Graphs”. In: *Transactions on Design Automation of Electronic Systems* 3.1 (1998), pp. 51–75. ISSN: 1084-4309

[236] S. Liao, S. Devadas, K. Keutzer, and S. Tjiang. “Instruction Selection Using Binate Covering for Code Size Optimization”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ICCAD’95. San Jose, California, USA: IEEE Computer Society, 1995, pp. 393–399. ISBN: 0-8186-7213-7

[237] C. Liem, T. May, and P. Paulin. “Instruction-Set Matching and Selection for DSP and ASIP Code Generation”. In: *Proceedings of European Design and Test Conference (EDAC/ETC/EUROASIC’94)*. Washington, District of Colombia, USA: IEEE Computer Society, 1994, pp. 31–37

[238] C. Lindig and N. Ramsey. *OCamlBURG*. 2006. URL: `http://www.cminusminus.org/` (visited on 2014-02-11)

[239] J. Liu, Y. Zhang, O. Jang, W. Ding, and M. Kandemir. “A Compiler Framework for Extracting Superword Level Parallelism”. In: *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’12. Beijing, China: ACM, 2012, pp. 347–358. ISBN: 978-1-4503-1205-9

[240] E. M. Loiola, N. M. Maia de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. “A Survey for the Quadratic Assignment Problem”. In: *European Journal of Operational Research* 176.2 (2007), pp. 657–690. ISSN: 0377-2217

[241] M. Lorenz, R. Leupers, P. Marwedel, T. Drager, and G. Fettweis. “Low-Energy DSP Code Generation Using a Genetic Algorithm”. In: *Proceedings of the International Conference on Cmoputer Design*. ICCD’01. Washington, District of Colombia, USA: IEEE Computer Society, 2001, pp. 431–437

[242] M. Lorenz and P. Marwedel. “Phase Coupled Code Generation for DSPs Using a Genetic Algorithm”. In: *Proceedings of the 9th Conference and Exhibition on Design, Automation and Test in Europe*. Vol. 2. DATE’04. Washington, District of Colombia, USA: IEEE Computer Society, Feb. 2004, pp. 1270–1275

[243] E. S. Lowry and C. W. Medlock. “Object Code Optimization”. In: *Communications of the ACM* 12.1 (1969), pp. 13–22. ISSN: 0001-0782

[244] H. Lunell. “Code Generator Writing Systems”. No. 94. Doctoral thesis. Linköping, Sweden: Linköping University, 1983

[245] M. Madhavan, P. Shankar, S. Rai, and U. Ramakrishna. “Extending Graham-Glanville Techniques for Optimal Code Generation”. In: *Transactions on Programming Languages and Systems* 22.6 (2000), pp. 973–1001. ISSN: 0164-0925

[246] M. Mahmood, F. Mavaddat, and M. Elmastry. “Experiments with an Efficient Heuristic Algorithm for Local Microcode Generation”. In: *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*. ICCD’90. Washington, District of Colombia, USA: IEEE Computer Society, 1990, pp. 319–323

[247] I. J. Maltz. “Implementation of a Code Generator Preprocessor”. MA thesis. Berkeley, California, USA: University of California, 1978

[248] K. Martin, C. Wolinski, K. Kuchcinski, A. Floch, and F. Charot. “Constraint Programming Approach to Reconfigurable Processor Extension Generation and Application Compilation”. In: *Transactions on Reconfigurable Technology and Systems* 5.2 (2012), 10:1–10:38. ISSN: 1936-7406

[249] K. Martin, C. Wolinski, K. Kuchcinski, A. Floch, and F. Charot. “Constraint-Driven Instructions Selection and Application Scheduling in the DURASE System”. In: *Proceedings of the 20th International Conference on Application-Specific Systems, Architectures and Processors*. ASAP’09. Washington, District of Colombia, USA: IEEE Computer Society, 2009, pp. 145–152. ISBN: 978-0-7695-3732-0

[250] P. Marwedel. “Code Generation for Core Processors”. In: *Proceedings of the Design Automation Conference*. DAC’97. Anaheim, California, USA: IEEE Computer Society, 1997, pp. 232–237. ISBN: 0-7803-4093-0

[251] P. Marwedel. “The MIMOLA Design System: Tools for the Design of Digital Processors”. In: *Proceedings of the 21st Design Automation Conference*. DAC’84. Albuquerque, New Mexico, USA: IEEE Computer Society, 1984, pp. 587–593. ISBN: 0-8186-0542-1

[252] P. Marwedel. “Tree-Based Mapping of Algorithms to Predefined Structures”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. ICCAD’93. Santa Clara, California, USA: IEEE Computer Society, 1993, pp. 586–593. ISBN: 0-8186-4490-7

[253] H. Massalin. “Superoptimizer: A Look at the Smallest Program”. In: *Proceedings of the 2nd International Conference on Architectual Support for Programming Languages and Operating Systems*. ASPLOS II. Palo Alto, California, USA: IEEE Computer Society, 1987, pp. 122–126. ISBN: 0-8186-0805-6

[254] W. M. McKeeman. “Peephole Optimization”. In: *Communications of the ACM* 8.7 (July 1965), pp. 443–444. ISSN: 0001-0782

[255] P. L. Miller. “Automatic Creation of a Code Generator from a Machine Description”. MA thesis. Cambridge, Massachusetts, USA: Massachusetts Institute of Technology, 1971

[256] S. Mouthuy, Y. Deville, and G. Dooms. “Global Constraint for the Set Covering Problem”. In: *Proceedings of the 3rd French-Speaking Conference on Constraint Programming*. JFPC’07. INRIA, Domaine de Voluceau, Rocquencourt, Yvelines, France, June 2007, pp. 183–192

[257] S. Muchnick. *Advanced Compiler Design & Implementation*. Burlington, Massachusetts, USA: Morgan Kaufmann, 1997. ISBN: 978-1558603202

[258] C. Müller. *Code Selection from Directed Acyclic Graphs in the Context of Domain Specific Digital Signal Processors*. Tech. rep. Berlin, Germany: Humboldt-Universität, 1994

[259] A. C. Murray. “Customising Compilers for Customisable Processors”. Doctoral thesis. Edinburgh, Scotland: University of Edinburgh, 2012

[260] A. Murray and B. Franke. “Compiling for Automatically Generated Instruction Set Extensions”. In: *Proceedings of the 10th International Symposium on Code Generation and Optimization*. CGO’12. San Jose, California, USA: ACM, 2012, pp. 13–22. ISBN: 978-1-4503-1206-6

[261] M. Mutyam, F. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. “Compiler-Directed Thermal Management for VLIW Functional Units”. In: *Proceedings of the SIGPLAN/SIGBED Conference on Language, Compilers, and Tool Support for Embedded Systems*. LCTES’06. Ottawa, Ontario, Canada: ACM, 2006, pp. 163–172. ISBN: 1-59593-362-X

[262] I. Nakata. “On Compiling Algorithms for Arithmetic Expressions”. In: *Communications of the ACM* 10.8 (1967), pp. 492–494. ISSN: 0001-0782

[263] J. M. Newcomer. “Machine-Independent Generation of Optimal Local Code”. Order number: AAI7521781. Doctoral thesis. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University, 1975

[264] A. Newell and H. A. Simon. *The Simulation of Human Thought*. Tech. rep. Santa Monica, California, USA: Mathematics Devision, RAND Corporation, June 1959

[265] A. Nicolau and S. Novack. “An Efficient Global Resource Constrained Technique for Exploiting Instruction Level Parallelism”. In: *Proceedings of the International Conference on Parallel Processing*. Ed. by K. G. Shin. Vol. 2. ICPP’92. University of Michigan, Ann Arbor, Michigan, USA, 1992, pp. 297–301

[266] R. Niemann and P. Marwedel. “An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming”. In: *Design Automation for Embedded Systems* 2.2 (1997), pp. 165–193. ISSN: 0929-5585

[267] S. Novack, A. Nicolau, and N. Dutt. “A Unified Code Generation Approach Using Mutation Scheduling”. In: *Code Generation for Embedded Processors*. Ed. by P. Marwedel and G. Goossens. Vol. 317. Springer, 2002. Chap. 12, pp. 203–218. ISBN: 978-1-4613-5983-8

[268] S. Novack and A. Nicolau. “Mutation Scheduling: A Unified Approach to Compiling for Fine-Grain Parallelism”. In: *Proceedings of the 7th International Workshop on Languages and Compilers for Parallel Computing*. LCPC’94. Springer, 1995, pp. 16–30. ISBN: 3-540-58868-X

[269] L. Nowak and P. Marwedel. “Verification of Hardware Descriptions by Retargetable Code Generation”. In: *Proceedings of the 26th ACM/IEEE Design Automation Conference*. DAC’89. Las Vegas, Nevada, USA: ACM, 1989, pp. 441–447. ISBN: 0-89791-310-8

[270] A. Nymeyer and J.-P. Katoen. “Code Generation Based on Formal Bottom-Up Rewrite Systems Theory and Heuristic Search”. In: *Acta Informatica* 34.4 (1997), pp. 597–635

[271] A. Nymeyer, J.-P. Katoen, Y. Westra, and H. Alblas. “Code Generation = A^* + BURS”. In: *Proceedings of the 6th International Conference on Compiler Construction (CC’06)*. Ed. by T. Gyimóthy. Vol. 1060. Lecture Notes in Computer Science. Springer, 1996, pp. 160–176. ISBN: 978-3-540-61053-3

[272] M. J. O’Donnell. *Equational Logic as a Programming Language*. Cambridge, Massachusetts, USA: MIT Press, 1985. ISBN: 978-0262150286

[273] R. J. Orgass and W. M. Waite. “A Base for a Mobile Programming System”. In: *Communications of the ACM* 12.9 (1969), pp. 507–510. ISSN: 0001-0782

[274] M. Paleczny, C. Vick, and C. Click. “The Java Hotspot™ Server Compiler”. In: *Proceedings of the Symposium on Java Virtual Machine Research and Technology Symposium*. JVM’01. Monterey, California, USA: USENIX Association, 2001

[275] A. Parikh, S. Kim, M. Kandemir, N. Vijaykrishnan, and M. Irwin. “Instruction Scheduling for Low Power”. In: *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology* 37.1 (2004), pp. 129–149. ISSN: 0922-5773

[276] P. G. Paulin, C. Liem, T. C. May, and S. Sutarwala. “DSP Design Tool Requirements for Embedded Systems: A Telecommunications Industrial Perspective”. In: *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology* 9.1–2 (1995), pp. 23–47. ISSN: 0922-5773

[277] P. P. Paulin, C. Liem, T. May, and S. Sutarwala. “CodeSyn: A Retargetable Code Synthesis System”. In: *Proceedings of the 7th International Symposium on High Level Synthesis*. San Diego, California, USA: IEEE Computer Society, 1994, pp. 94–95

[278] E. Pelegrí-Llopart and S. L. Graham. “Optimal Code Generation for Expression Trees: An Application of BURS Theory”. In: *Proceedings of the 15th SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’88. San Diego, California, USA: ACM, 1988, pp. 294–308. ISBN: 0-89791-252-7

[279] T. J. Pennello. “Very Fast LR Parsing”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’86. Palo Alto, California, USA: ACM, 1986, pp. 145–151. ISBN: 0-89791-197-0

[280] D. R. Perkins and R. L. Sites. “Machine-Independent PASCAL Code Optimization”. In: *Proceedings of the SIGPLAN Symposium on Compiler Construction*. SIGPLAN’79. Denver, Colorado, USA: ACM, 1979, pp. 201–207. ISBN: 0-89791-002-8

[281] T. A. Proebsting. “BURS Automata Generation”. In: *Transactions on Programming Language Systems* 17.3 (1995), pp. 461–486. ISSN: 0164-0925

[282] T. A. Proebsting. “Code Generation Techniques”. #1119. Doctoral thesis. Madison, Wisconsin, USA: The University of Wisconsin–Madison, Nov. 1992

[283] T. A. Proebsting. *Least-Cost Instruction Selection in DAGs is NP-Complete*. 1995. URL: `http://web.archive.org/web/20081012050644/http://research.microsoft.com/~toddpro/papers/proof.htm` (visited on 2013-04-23)

[284] T. A. Proebsting. “Simple and Efficient BURS Table Generation”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’92. San Francisco, California, USA: ACM, 1992, pp. 331–340. ISBN: 0-89791-475-9

[285] T. A. Proebsting and B. R. Whaley. “One-Pass, Optimal Tree Parsing—With or Without Trees”. In: *Proceedings of the 6th International Conference on Compiler Construction (CC’06)*. Ed. by T. Gyimóthy. Vol. 1060. Lecture Notes in Computer Science. Springer, 1996, pp. 294–306. ISBN: 978-3-540-61053-3

[286] P. W. Purdom Jr. and C. A. Brown. “Fast Many-to-One Matching Algorithms”. In: *Proceedings of the 1st International Conference on Rewriting Techniques and Applications*. Dijon, France: Springer, 1985, pp. 407–416. ISBN: 0-387-15976-2

[287] R. Ramesh and I. V. Ramakrishnan. “Nonlinear Pattern Matching in Trees”. In: *Journal of the ACM* 39.2 (1992), pp. 295–316. ISSN: 0004-5411

[288] N. Ramsey and J. W. Davidson. “Machine Descriptions to Build Tools for Embedded Systems”. In: *Proceedings of the SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*. LCTES’98. Springer, 1998, pp. 176–192. ISBN: 3-540-65075-X

[289] N. Ramsey and J. Dias. “Resourceable, Retargetable, Modular Instruction Selection Using a Machine-Independent, Type-Based Tiling of Low-Level Intermediate Code”. In: *Proceedings of the 38th Annual SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’11. Austin, Texas, USA: ACM, 2011, pp. 575–586. ISBN: 978-1-4503-0490-0

[290] B. R. Rau and J. A. Fisher. “Instruction-Level Parallel Processing: History, Overview, and Perspective”. In: *Journal of Supercomputing* 7.1–2 (May 1993), pp. 9–50. ISSN: 0920-8542

[291] R. R. Redziejowski. “On Arithmetic Expressions and Trees”. In: *Communications of the ACM* 12.2 (1969), pp. 81–84. ISSN: 0001-0782

[292] C. R. Reeves. “Genetic Algorithms”. In: *Handbook of Metaheuristics*. Ed. by M. Gendreau and J.-Y. Potvin. 2nd ed. Vol. 146. International Series in Operations Research & Management Science. Springer, 2010. Chap. 5, pp. 109–139. ISBN: 978-1-4419-116-1

[293] J. F. Reiser. “Compiling Three-Address Code for C Programs”. In: *The Bell System Technical Journal* 60.2 (1981), pp. 159–166

[294] K. Ripken. *Formale Beschreibung von Maschinen, Implementierungen und Optimierender Maschinencodeerzeugung aus Attributierten Programmgraphen*. Tech. rep. TUM-INFO-7731. Munich, Germany: Institut für Informatik, Technical University of Munich, July 1977

[295] F. Rossi, P. Van Beek, and T. Walsh, eds. *Handbook of Constraint Programming*. Amsterdam, Netherlands: Elsevier, 2006. ISBN: 978-0-444-52726-4

[296] R. L. Rudell. “Logic Synthesis for VLSI Design”. AAI9006491. Doctoral thesis. Berkeley, California, USA: University of California, 1989

[297] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. London, England: Pearson Education, 2010. ISBN: 0-13-604259-7

[298] E. D. Sacerdoti. “Planning in a Hierarchy of Abstraction Spaces”. In: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*. Ed. by N. J. Nilsson. IJCAI’73. Stanford, California, USA: Morgan Kaufmann, 1973, pp. 412–422

[299] S. Sakai, M. Togasaki, and K. Yamazaki. “A Note on Greedy Algorithms for the Maximum Weghted Independent Set Problem”. In: *Discrete Applied Mathematics* 126.2-3 (2003), pp. 313–322. ISSN: 0166-218X

[300] V. Sarkar, M. J. Serrano, and B. B. Simons. “Register-Sensitive Selection, Duplication, and Sequencing of Instructions”. In: *Proceedings of the 15th International Conference on Supercomputing*. ICS’01. Sorrento, Italy: ACM, 2001, pp. 277–288. ISBN: 1-58113-410-X

[301] B. Schafer, Y. Lee, and T. Kim. “Temperature-Aware Compilation for VLIW Processors”. In: *Proceedings for the 13th International Conference on Embedded and Real-Time Computing Systems and Applications*. RTCSA’07. Daegu, Korea: IEEE Computer Society, Aug. 2007, pp. 426–431

[302] S. Schäfer and B. Scholz. “Optimal Chain Rule Placement for Instruction Selection Based on SSA Graphs”. In: *Proceedings of the 10th International Workshop on Software and Compilers for Embedded Systems*. SCOPES’07. Nice, France: ACM, 2007, pp. 91–100

[303] H. Scharwaechter, J. M. Youn, R. Leupers, Y. Paek, G. Ascheid, and H. Meyr. “A Code-Generator Generator for Multi-Output Instructions”. In: *Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS’07. Salzburg, Austria: ACM, 2007, pp. 131–136. ISBN: 978-1-59593-824-4

[304] B. Scholz and E. Eckstein. “Register Allocation for Irregular Architectures”. In: *Proceedings of the joint Conference on Languages, Compilers and Tools for Embedded Systems and Software and Compilers for Embedded Systems*. LCTES/SCOPES’02. Berlin, Germany: ACM, 2002, pp. 139–148. ISBN: 1-58113-527-0

[305] R. Sethi and J. D. Ullman. “The Generation of Optimal Code for Arithmetic Expressions”. In: *Journal of the ACM* 17.4 (1970), pp. 715–28. ISSN: 0004-5411

[306] R. Shamir and D. Tsur. “Faster Subtree Isomorphism”. In: *Journal of Algorithms* 33.2 (1999), pp. 267–280. ISSN: 0196-6774

[307] P. Shankar, A. Gantait, A. R. Yuvaraj, and M. Madhavan. “A New Algorithm for Linear Regular Tree Pattern Matching”. In: *Theoretical Computer Science* 242.1-2 (2000), pp. 125–142. ISSN: 0304-3975

[308] J. Shu, T. C. Wilson, and D. K. Banerji. “Instruction-Set Matching and GA-based Selection for Embedded-Processor Code Generation”. In: *Proceedings of the 9th International Conference on VLSI Design: VLSI in Mobile Communication*. VLSID’96. Washington, District of Colombia, USA: IEEE Computer Society, 1996, pp. 73–76. ISBN: 0-8186-7228-5

[309] D. C. Simoneaux. “High-Level Language Compiling for User-Defineable Architectures”. Doctoral thesis. Monterey, California, USA: Naval Postgraduate School, 1975

[310] A. Snyder. “A Portable Compiler for the Language C”. MA thesis. Cambridge, Massachusetts, USA, 1975

[311] S. Sorlin and C. Solnon. “A Global Constraint for Graph Isomorphism Problems”. In: *Proceedings of the 1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR’04)*. Ed. by J.-C. Régin and M. Rueher. Vol. 3011. Lecture Notes in Computer Science. Springer, 2004, pp. 287–301. ISBN: 978-3-540-21836-4

[312] V. Srinivasan and T. Reps. “Synthesis of Machine Code from Semantics”. In: *Proceedings of the 36th SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’15. Portland, OR, USA: ACM, 2015, pp. 596–607. ISBN: 978-1-4503-3468-6

[313] R. Stallman. *Internals of GNU CC*. Version 1.21. Free Software Foundation, Inc. Apr. 24, 1988. URL: `http://trinity.engr.uconn.edu/~vamsik/internals.pdf` (visited on 2013-05-29)

[314] J. Stanier and D. Watson. “Intermediate Representations in Imperative Compilers: A Survey”. In: *Computing Surveys* 45.3 (July 2013), 26:1–26:27. ISSN: 0360-0300

[315] J. Strong, J. Wegstein, A. Tritter, J. Olsztyn, O. Mock, and T. Steel. “The Problem of Programming Communication with Changing Machines: A Proposed Solution”. In: *Communications of the ACM* 1.8 (1958), pp. 12–18. ISSN: 0001-0782

[316] A. Sudarsanam, S. Malik, and M. Fujita. “A Retargetable Compilation Methodology for Embedded Digital Signal Processors Using a Machine-Dependent Code Optimization Library”. In: *Design Automation for Embedded Systems* 4.2–3 (1999), pp. 187–206. ISSN: 0929-5585

[317] H. Tanaka, S. Kobayashi, Y. Takeuchi, K. Sakanushi, and M. Imai. “A Code Selection Method for SIMD Processors with PACK Instructions”. In: *Proceedings of the 7th International Workshop on Software and Compilers for Embedded Systems (SCOPES’03)*. Ed. by A. Krall. Vol. 2826. Lecture Notes in Computer Science. Springer, 2003, pp. 66–80. ISBN: 978-3-540-20145-8

[318] A. S. Tanenbaum, H. van Staveren, E. G. Keizer, and J. W. Stevenson. “A Practical Tool Kit for Making Portable Compilers”. In: *Communications of the ACM* 26.9 (1983), pp. 654–660. ISSN: 0001-0782

[319] A. K. Tirrell. “A Study of the Application of Compiler Techniques to the Generation of Micro-Code”. In: *Proceedings of the SIGPLAN/SIGMICRO Interface Meeting*. Harriman, New York, USA: ACM, 1973, pp. 67–85

[320] S. W. K. Tjiang. *An Olive Twig*. Tech. rep. Synopsys Inc., 1993

[321] S. W. K. Tjiang. *Twig Reference Manual*. Tech. rep. Murray Hill, New Jersey, USA: AT&T Bell Laboratories, 1986

[322] *TMS320C55x DSP Mnemonic Instruction Set Reference Guide*. SPRU374G. Texas Instruments. Oct. 2002

[323] K. Trifunovic, D. Nuzman, A. Cohen, A. Zaks, and I. Rosen. “Polyhedral-Model Guided Loop-Nest Auto-Vectorization”. In: *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*. PACT’09. Washington, District of Colombia, USA: IEEE Computer Society, 2009, pp. 327–337. ISBN: 978-0-7695-3771-9

[324] J. R. Ullmann. “An Algorithm for Subgraph Isomorphism”. In: *Journal of the ACM* 23.1 (1976), pp. 31–42. ISSN: 0004-5411

[325] P. Van Beek. Private correspondence. Nov. 2014

[326] J. Van Praet, D. Lanneer, W. Geurts, and G. Goossens. “Processor Modeling and Code Selection for Retargetable Compilation”. In: *Transactions on Design Automation of Electronic Systems* 6.3 (2001), pp. 277–307. ISSN: 1084-4309

[327] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man. “Instruction Set Definition and Instruction Selection for ASIPs”. In: *Proceedings of the 7th International Symposium on Systems Synthesis*. ISSS’94. Niagara-on-the-Lake, Ontario, Canada: IEEE Computer Society, 1994, pp. 11–16. ISBN: 0-8186-5785-5

[328] B.-S. Visser. “A Framework for Retargetable Code Generation Using Simulated Annealing”. In: *Proceedings of the 25th EUROMICRO’99 Conference on Informatics: Theory and Practice for the New Millennium*. Washington, District of Colombia, USA: IEEE Computer Society, 1999, pp. 1458–1462. ISBN: 0-7695-0321-7

[329] E. Visser. “A Survey of Strategies in Rule-Based Program Transformation Systems”. In: *Journal of Symbolic Computation* 40.1 (2005), pp. 831–873

[330] E. Visser. “Stratego: A Language for Program Transformation Based on Rewriting Strategies - System Description of Stratego 0.5”. In: *Rewriting Techniques and Applications*. Vol. 2051. RTA’01. Springer, 2001, pp. 357–361

[331] S. G. Wasilew. “A Compiler Writing System with Optimization Capabilities for Complex Object Order Structures”. AAI7232604. Doctoral thesis. Evanston, Illinois, USA: Northwestern University, 1972

[332] S. W. Weingart. “An Efficient and Systematic Method of Compiler Code Generation”. AAI7329501. Doctoral thesis. New Haven, Connecticut, USA: Yale University, 1973

[333] D. Weise, R. F. Crew, M. Ernst, and B. Steensgaard. “Value Dependence Graphs: Representation Without Taxation”. In: *Proceedings of the 21st SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL’94. Portland, Oregon, USA: ACM, 1994, pp. 297–310. ISBN: 0-89791-636-0

[334] B. Weisgerber and R. Wilhelm. “Two Tree Pattern Matchers for Code Selection”. In: *Proceedings of the 2nd CCHSC Workshop on Compiler Compilers and High Speed Compilation*. Springer, 1989, pp. 215–229. ISBN: 3-540-51364-7

[335] A. L. Wendt. “Fast Code Generation Using Automatically-Generated Decision Trees”. In: *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*. PLDI’90. White Plains, New York, USA: ACM, 1990, pp. 9–15. ISBN: 0-89791-364-7

[336] B. Wess. “Automatic Instruction Code Generation Based on Trellis Diagrams”. In: *Proceedings of the International Symposium on Circuits and Systems*. Vol. 2. ISCAS’92. Washington, District of Colombia, USA: IEEE Computer Society, 1992, pp. 645–648

[337] B. Wess. “Code Generation Based on Trellis Diagrams”. In: *Code Generation for Embedded Processors*. Ed. by P. Marwedel and G. Goossens. Springer, 1995. Chap. 11, pp. 188–202. ISBN: 978-0-7923-9577-5

[338] T. R. Wilcox. “Generating Machine Code for High-Level Programming Languages”. AAI7209959. Doctoral thesis. Ithaca, New York, USA: Cornell University, 1971

[339] R. Wilhelm and D. Maurer. *Compiler Design*. Boston, Massachusetts, USA: Addison-Wesley, 1995. ISBN: 978-0201422-900

[340] T. Wilson, G. Grewal, B. Halley, and D. Banerji. “An Integrated Approach to Retargetable Code Generation”. In: *Proceedings of the 7th International Symposium on High-Level Synthesis*. ISSS’94. Niagara-on-the-Lake, Ontario, Canada: IEEE Computer Society, 1994, pp. 70–75. ISBN: 0-8186-5785-5

[341] L. A. Wolsey. *Integer Programming*. Hoboken, New Jersey, USA: Wiley, 1998

[342] P. Wu, A. E. Eichenberger, and A. Wang. “Efficient SIMD Code Generation for Runtime Alignment and Length Conversion”. In: *Proceedings of the International Symposium on Code Generation and Optimization*. CGO’05. Washington, District of Colombia, USA: IEEE Computer Society, 2005, pp. 153–164. ISBN: 0-7695-2298-X

[343] S. Wu and S. Li. “Instruction Selection for ARM/Thumb Processors Based on a Multi-objective Ant Algorithm”. In: *Computer Science—Theory and Applications*. Ed. by D. Grigoriev, J. Harrison, and E. A. Hirsch. Vol. 3967. Lecture Notes in Computer Science. Springer, 2006, pp. 641–651. ISBN: 978-3-540-34166-6

[344] W. A. Wulf, R. K. Johnsson, C. B. Weinstock, S. O. Hobbs, and C. M. Geschke. *The Design of an Optimizing Compiler*. Amsterdam, Netherlands: Elsevier, 1975. ISBN: 0444001581

[345] H.-T. L. Wuu and W. Yang. “A Simple Tree Pattern-Matching Algorithm”. In: *Proceedings of the Workshop on Algorithms and Theory of Computation*. Chiyayi, Taiwan, 2000, pp. 1–8

[346] M. Xie, C. Pan, J. Hu, C. Xue, and Q. Zhuge. “Non-Volatile Registers Aware Instruction Selection for Embedded Systems”. In: *Proceedings of the 20th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. RTCSA’14. Washington, District of Colombia, USA: IEEE Computer Society, Aug. 2014, pp. 1–9

[347] W. Yang. “A Fast General Parser for Automatic Code Generation”. In: *Proceedings of the 2nd Russia-Taiwan Conference on Methods and Tools of Parallel Programming Multicomputers*. MTPP’10. Vladivostok, Russia: Springer, 2010, pp. 30–39. ISBN: 978-3-642-14821-7

[348] J. S. Yates and R. A. Schwartz. “Dynamic Programming and Industrial-Strength Instruction Selection: Code Generation by Tiring, but not Exhaustive, Search”. In: *SIGPLAN Notices* 23.10 (1988), pp. 131–140. ISSN: 0362-1340

[349] J. M. Youn, J. Lee, Y. Paek, J. Lee, H. Scharwaechter, and R. Leupers. “Fast Graph-Based Instruction Selection for Multi-Output Instructions”. In: *Software—Practice & Experience* 41.6 (2011), pp. 717–736. ISSN: 0038-0644

[350] R. Young. “The Coder: A Program Module for Code Generation in High Level Language Compilers”. MA thesis. Urbana-Champaign, Illinois, USA: Computer Science Department, University of Illinois, 1974

[351] K. H. Yu and Y. H. Hu. “Artificial Intelligence in Scheduling and Instruction Selection for Digital Signal Processors”. In: *Applied Artificial Intelligence* 8.3 (1994), pp. 377–392

[352] K. H. Yu and Y. H. Hu. “Efficient Scheduling and Instruction Selection for Programmable Digital Signal Processors”. In: *Transactions on Signal Processing* 42.12 (1994), pp. 3549–3552. ISSN: 1053-587X

[353] P. Yu and T. Mitra. “Scalable Custom Instructions Identification for Instruction-Set Extensible Processors”. In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. CASES’04. Washington, District of Colombia, USA: ACM, 2004, pp. 69–78

[354] G. Zimmermann. “The MIMOLA Design System: A Computer Aided Digital Processor Design Method”. In: *Proceedings of the 16th Design Automation Conference*. DAC’79. San Diego, California, USA: IEEE Computer Society, 1979, pp. 53–58

[355] W. Zimmermann and T. Gaul. “On the Construction of Correct Compiler Back-Ends: An ASM-Approach”. In: *Journal of Universal Computer Science* 3.5 (May 28, 1997), pp. 504–567

[356] J. Ziv and A. Lempel. “A Universal Algorithm for Sequential Data Compression”. In: *Transactions on Information Theory* 23.3 (1977), pp. 337–343

[357] V. Živojnović, J. Martínez Velarde, C. Schläger, and H. Meyr. “DSPstone: A DSP-Oriented Benchmarking Methodology”. In: *Proceedings of the International Conference on Signal Processing Applications and Technology*. ICSPAT’94. Dallas, Texas, USA, Oct. 1994, pp. 715–720

Index

Symbols

A

B

© Springer International Publishing Switzerland 2016

G. Hjort Blindell, *Instruction Selection*,
DOI 10.1007/978-3-319-34019-7

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z