VLIW DSP Instruction Clustering Scheduling Based on Integer Linear Programming (Post-print)

Authors: Zhou Peng, Liu Chungang, Zheng Qilong

Date: 2022-06-06T00:00:00+00:00

Abstract

On clustered VLIW DSPs, instruction clustering is a compiler optimization that significantly impacts program performance, but existing instruction clustering algorithms can only handle sequential program regions and struggle to obtain optimal clustering solutions. To address these issues, this paper proposes a unified method for instruction clustering and scheduling based on integer linear programming. This method employs 0-1 decision variables to represent the clustering of instructions within a function, the local scheduling of instructions, and the global scheduling of inter-cluster transfer instructions, and constructs linear constraints from the dependency relations between instructions and contention for processor resources, ultimately yielding an integer linear programming model with the objective of minimizing the estimated execution time of the function. Experimental results demonstrate that the clustering and scheduling scheme obtained by solving this model significantly outperforms existing algorithms in optimizing program performance, and the computational time required to solve the model is acceptable.

Full Text

Preamble

Vol. 39 No. 10
Application Research of Computers
ChinaXiv Cooperative Journal

Integer Linear Programming Based Instruction Cluster Assignment and Schedule of VLIW DSP

Zhou Peng[1],[2], Liu Chungang[1],[2], Zheng Qilong[1],[2]

[1](University of Science & Technology of China, College of Computer Science & Technology, Hefei 230026, China;
[2](Key Laboratory of High Performance Computing, Anhui Province, Hefei 230026, China)

Abstract: On clustered VLIW DSPs, instruction cluster assignment is a crucial compiler optimization that significantly impacts program performance. However, existing cluster assignment algorithms are limited to straight-line program regions and struggle to achieve optimal solutions. To address these limitations, this paper proposes a unified approach for instruction cluster assignment and scheduling based on integer linear programming. The method employs zero-one decision variables to represent cluster assignment for each instruction, local instruction scheduling within basic blocks, and global scheduling of inter-cluster transfer instructions. Instruction dependencies and resource contention on the processor are formulated as linear constraints, culminating in an integer linear programming model that minimizes the estimated execution time of the function. Experimental results demonstrate that the cluster assignment and scheduling solutions obtained from solving this model substantially outperform existing algorithms in program performance optimization, while the time required to solve the model remains acceptable.

Key words: DSP; VLIW; cluster assignment; instruction scheduling; integer linear programming

0 Introduction

Many digital signal processors (DSPs) adopt Very Long Instruction Word (VLIW) architectures to exploit instruction-level parallelism (ILP). VLIW DSPs typically feature large issue widths, requiring numerous functional units and register files. Directly connecting all functional units to all registers would result in high design complexity and excessive power consumption. A common solution partitions functional units and registers into multiple clusters, where each cluster contains an equal number of functional units and registers. Functional units within a cluster can directly access the local register file, while inter-cluster communication occurs via dedicated buses. Accessing data across clusters requires copying data from remote clusters to the local cluster. In such clustered architectures, intra-cluster data access incurs low latency and power consumption, whereas inter-cluster access suffers from higher latency and power costs.

On clustered architectures, determining which cluster' s functional resources each instruction will use is called cluster assignment. In some early processors, cluster assignment was performed automatically by hardware, as described in the literature. In modern clustered VLIW DSPs, such as the HXDSP developed by the 38th Research Institute of China Electronics Technology Group, cluster assignment is handled by the compiler. The goal of cluster assignment is to

fully utilize computational resources across different clusters while minimizing inter-cluster communication, thereby improving ILP. This makes cluster assignment intimately related to instruction scheduling. Early approaches treated cluster assignment as a separate optimization pass performed before instruction scheduling, while more recent work performs cluster assignment and scheduling simultaneously.

Current cluster assignment methods operate on straight-line program regions such as basic blocks or traces. These methods differ primarily in their heuristic factors for determining cluster assignment. However, they inadequately handle global data flow, neglecting inter-cluster communication costs between instructions in different basic blocks. Although trace-based assignment considers multiple basic blocks, constructing effective traces is difficult in many programs due to unpredictable branch behavior. Another fundamental limitation is that heuristic-based approaches cannot guarantee optimal solutions. While heuristics offer fast execution, in DSP application domains, spending more compilation time to achieve better cluster assignments and higher execution performance is often acceptable.

Combinatorial optimization provides methods for finding optimal solutions, among which integer linear programming (ILP) stands out for its simple formulation yet wide applicability in compiler optimization, including instruction scheduling and superword-level parallelism. Given the suboptimal nature of existing heuristic cluster assignment algorithms, employing ILP represents a promising approach to further improve cluster assignment quality.

The main contributions of this paper are: (1) We propose an ILP modeling approach for unified instruction cluster assignment and scheduling on VLIW DSPs. The model represents cluster assignment for every instruction in a function, instruction scheduling within each basic block, and the insertion positions and scheduling of inter-cluster transfer instructions resulting from cluster assignment, all within a single ILP framework. (2) Compared to existing methods, our approach performs cluster assignment at the function level, addressing the insertion of inter-cluster transfer instructions across global data flow, and yields optimal cluster assignment and scheduling solutions (under the estimated cost model). (3) The method can be directly applied to other program regions, such as loop bodies, basic blocks, or traces. When applied to straight-line code regions, execution time can be accurately evaluated, yielding truly optimal solutions.

The earliest instruction cluster assignment method was the BUG (Bottom-Up Greedy) algorithm implemented in the Bulldog compiler. BUG uses a heuristic called completion cycle, defined as the latest time when an instruction completes execution on a candidate cluster and its result is transferred to the cluster where its flow-dependent successor resides. Starting from instructions without successors, BUG traverses the data flow graph in reverse depth-first order, selecting the cluster that minimizes the completion cycle for each instruction, then assigns all its predecessors before finally assigning the current instruction to a

cluster with minimal completion cycle.

BUG' s greedy approach cannot guarantee that all instructions on the critical path are assigned to the same cluster. Moreover, minimizing completion time during the initial depth-first traversal may scatter data-dependent instructions across different clusters. Lowney et al. improved upon BUG by partitioning the data flow graph into components with low parallelism but high data sharing. When calculating completion cycles, they imposed a large penalty if two related components resided in different clusters, encouraging related components to be placed in the same cluster and avoiding inter-cluster communication.

Desoli et al. employed a multi-phase iterative algorithm for cluster assignment. They first obtained an initial clustering using a BUG-like algorithm that minimized inter-cluster communication, then iteratively performed list scheduling, adjusting the clustering based on execution cycles, register pressure, and inter-cluster communication until convergence. List scheduling is a widely used instruction scheduling framework that performs a prioritized topological sort of the data dependence graph based on heuristics such as instruction height or number of successors, then schedules each instruction as early as possible according to this order.

Lapinskii et al. sorted instructions by latest scheduling time, slack (difference between latest and earliest scheduling times), and number of data flow successors. They visited each instruction in this order, assigning it to the cluster that minimized a weighted sum of functional unit usage cost, bus usage cost, and inter-cluster communication cost. After initial clustering, they iteratively "perturbed" border instructions (those communicating with other clusters) to test whether reassigning them to other clusters could yield better results.

Ozer et al. first proposed a unified approach to instruction cluster assignment and scheduling called UAS (Unified Assignment and Scheduling). Operating within the list scheduling framework, UAS determines the cluster for the current ready instruction based on the completion times of its data flow predecessors (called completion-weighted predecessors, or CWP), then schedules the instruction and any required inter-cluster transfer instructions as early as possible.

Kailas et al. proposed the CARS code generation framework that unified instruction cluster assignment, scheduling, and register allocation based on list scheduling, still using start time as the clustering heuristic.

Zhang et al. introduced a sophisticated heuristic called lmax-successor-tree-consistent deadline, defined as the latest execution completion time of an instruction among all feasible cluster assignment and scheduling schemes for that instruction and all its successors (where each instruction' s issue time does not exceed a given bound). Within the list scheduling framework, they selected the ready instruction with the smallest heuristic value and assigned it to the cluster that enabled earliest issue.

Porpodas et al. proposed the LUCAS algorithm, which combines start time

and completion time heuristics. Within the list scheduling framework, LUCAS chooses the cluster that minimizes start time when facing congestion (number of ready instructions > issue width × inter-cluster communication latency) or high slack (slack > issue width × 2 × (inter-cluster communication latency - 1)), otherwise it selects the cluster that minimizes completion time.

Wang et al. used simulated annealing for cluster assignment and scheduling, using scheduled clock cycles as feedback to iteratively improve clustering quality. Some work integrates cluster assignment with modulo scheduling for loops, considering instruction clustering during software pipelining. While effective, these methods require loops to be branch-free or have simple branch patterns that can be converted to predicated instructions via If-Conversion, limiting their applicability.

1 Overview

Our method is implemented after instruction selection and before register allocation in the compiler. It takes an entire function as input, where the function is represented as a control flow graph and each basic block is represented as a data dependence graph.

Integer linear programming refers to linear programming where variables take integer values, defined as: given a series of integer variables, called decision variables, satisfying a series of inequalities, minimize a linear function, where is a real matrix and is a real vector. Zero-one programming, where decision variables only take values 0 or 1, is a common modeling approach. In this paper, both instruction cluster assignment and scheduling are modeled using zero-one programming. The main decision variables and related symbols are shown in Table 1 . The constraints can be roughly divided into three categories: instruction occurrence constraints, resource constraints, and dependency constraints. The optimization objective is the weighted sum of basic block execution times, where the weight is the estimated execution count of the basic block, and the basic block execution time is represented by the issue time of its last instruction.

2 Inter-Cluster Transfer Instructions

Consider any variable v. It may appear in different clusters depending on which cluster the instruction defining it is assigned to. Instructions using v may also be assigned to other clusters, necessitating inter-cluster transfer instructions to propagate v. For each possible cluster pair (m_1, m_2), there exists a corresponding inter-cluster transfer instruction x ^{m_1,m_2}. These transfer instructions may be scheduled in any basic block along any path between the definition and use of v. To represent these scheduling possibilities, decision variables are defined for each transfer instruction in every basic block along these paths.

For example, in the control flow graph shown in Figure 1 [Figure 1: see original paper], the paths between definition and use of variable v include A→E,

A→E→F, B→C→E, B→C→E→F, and B→D→F. Therefore, decision variables for inter-cluster transfer instructions related to v should be defined in basic blocks A, B, C, D, E, and F to express all possible insertion schemes. Figures 2 [Figure 2: see original paper] through 4 [Figure 4: see original paper] illustrate several cluster assignment and inter-cluster transfer instruction insertion schemes (inter-cluster transfer instructions are shown as X = Y , indicating transfer of v from cluster Y to X), demonstrating that transfer instructions can appear in any basic block.

However, scheduling inter-cluster transfer instructions only in existing basic blocks is not always optimal. Consider the control flow graph in Figure 5 [Figure 5: see original paper]. When I1 and I4 are assigned to different clusters, inserting the transfer instruction in block A causes redundant instruction execution along path A→C (Figure 6 [Figure 6: see original paper]), while inserting it in block D causes data errors along path B→D (Figure 7 [Figure 7: see original paper]). The solution is to insert the transfer instruction in a newly created basic block along path A→D (Figure 8 [Figure 8: see original paper]), ensuring it executes only when that specific path is taken. Although this may introduce additional jump instructions with longer latency, the overhead of repeatedly executing useless instructions could be greater when many transfers are involved. The trade-off between redundant execution and additional jump overhead is optimally resolved during ILP solving, but the solution space must contain such options.

Therefore, for any control flow graph edge, if its source has multiple outgoing edges or its target has multiple incoming edges, a new basic block must be constructed on that edge. If the edge is a fall-through edge (consecutive basic block addresses), the inserted block is empty; otherwise, it contains a jump instruction targeting the edge' s successor. Whether this jump appears in the final code depends on whether the new block contains inter-cluster transfer instructions. If the source has only one outgoing edge or the target has only one incoming edge, transfer instructions can be inserted in the source or target block, as inserting them in a new block would only increase overhead.

3.1 Instruction Occurrence Constraints

Fundamentally, each instruction to be clustered must be assigned to exactly one cluster. For any instruction i, the following constraint applies:

This ensures that the cluster number assigned to instruction i can be expressed as:

Similarly, any original instruction i in basic block B must be scheduled at exactly one clock cycle:

This allows the scheduled clock cycle of instruction i in basic block B to be expressed as:

Inter-cluster transfer instruction scheduling depends on the cluster assignment

of instructions that define and use the corresponding variable, summarized in four rules:

For any variable v, when any two instructions i and j that read or write v are assigned to different clusters m_1 and m_2, the inter-cluster transfer instruction x ^{m_1,m_2} must be scheduled in some basic block along every path between them. This constraint can be expressed as:

This rule ensures transfer instructions always read data from the cluster where the defining instruction resides, excluding "relay" transfer scenarios. Although relay schemes are legal, they cannot be uniquely optimal on most clustered VLIW DSPs. From a data flow perspective, the earliest issue time for a transfer instruction reading from another transfer instruction is always later than reading directly from the defining instruction' s cluster. Since inter-cluster transfer instructions only occupy the inter-cluster bus resource and do not compete with other instructions for resources, having all transfer instructions read directly from the defining instruction' s cluster always yields results at least as good as relay schemes. Therefore, this constraint does not limit clustering quality.

For any variable v with defining instruction i, along any path P from i to an instruction using v, for any cluster m_2, the inter-cluster transfer instruction x ^{m_1,m_2} is allowed to appear only when instruction i is not assigned to cluster m_2, and at most once in some basic block at some time:

For any basic block B on a path between definition and use of variable v, the transfer instruction x ^{m_1,m_2} is allowed only when the using instruction j is assigned to cluster m_2:

In any newly inserted basic block B, only inter-cluster transfer instructions and a jump instruction j_B may exist. When other transfer instructions are scheduled, the jump instruction must also be scheduled in B:

Conversely, if B contains no transfer instructions, it can be entirely removed. This rule need not be expressed as a constraint, as an extra jump instruction cannot be optimal and will be eliminated during solving.

3.2 Dependency Constraints

Dependencies between instructions in a basic block include three data dependencies: flow dependence, anti-dependence, and output dependence. Additionally, all other instructions cannot execute after a jump instruction (if present). For simplicity, this dependency is referred to as control dependence in this paper, though the term "control dependence" has a different meaning in widespread usage.

Dependency constraints require that the difference in issue times between two dependent instructions must be greater than or equal to the dependency latency. For dependencies in basic block B that are independent of intra-cluster registers or are control dependencies, the constraint can be uniformly expressed as:

where L represents dependency latency. In VLIW DSPs, control and anti-dependencies typically have latency 0, output dependencies have latency 1, and flow dependencies depend on pipeline length.

When anti- or output dependencies involve intra-cluster register passing, the dependency is naturally eliminated if the predecessor or successor is assigned to different clusters, allowing arbitrary scheduling. Otherwise, the latency constraint must be satisfied. Both cases can be unified using the big-M method:

The variable δ_{i,j} indicating whether instructions i and j are assigned to the same cluster requires additional constraints:

Flow dependencies involving inter-cluster transfer instructions require special consideration. For instructions i and j that define and use variable v respectively, for any clusters m_1 and m_2, only when transfer instruction x ^{m_1,m_2} is scheduled in the same basic block as j and i is assigned to cluster m_1 does j flow-depend on i:

Similarly, in any basic block B on a path between definition and use of variable v, the transfer instruction x ^{m_1,m_2} is allowed only when the defining instruction i is assigned to cluster m_1:

When i and j are in the same basic block and assigned to the same cluster, they have a flow dependence requiring constraints similar to Equation 13 with flow dependency latency. If assigned to different clusters, inter-cluster transfer instructions exist, and constraints 16 and 17 ensure their latency requirements are satisfied.

3.3 Resource Constraints

Legal instruction scheduling must satisfy that for any processor resource (e.g., ALU, immediate channel), the total usage by instructions executing at any time cannot exceed machine limits. The number of resource r used by instruction i in pipeline stage k is denoted RES_{i,k,r}. Resource conflicts typically occur only in a few pipeline stages, and RES_{i,k,r} is only defined for those conflicting stages.

Resources in clustered DSPs can be divided into two categories: global resources shared by the entire DSP (usage independent of instruction cluster) and cluster-local resources usable only by instructions in the same cluster (each cluster has identical quantities). Notably, inter-cluster data buses are cluster-local resources, represented as several read/write ports per cluster.

For scheduling in basic block B, for global resource r at any time t, the total resource usage by all executing instructions cannot exceed the machine limit:

For cluster-local resources other than bus ports, resource usage depends on the cluster assignment of instructions. The resource constraint is:

where represents logical conjunction of zero-one variables, implemented by introducing an additional zero-one variable with constraints.

For inter-cluster bus read ports (cluster-local resource), only transfer instructions reading from cluster m occupy the resource. Let X_B^{m,read} be the set of such transfer instructions possibly scheduled in B. The constraint is:

Similarly, for bus write ports, let X_B^{m,write} be the set of transfer instructions targeting cluster m possibly scheduled in B:

4 Optimization Objective

The goal of cluster assignment and scheduling is to minimize function execution time, which is undecidable and must be approximated. Under the constraints described above, the last instruction issue time in basic block B is:

The actual issue time range for each instruction in B is typically much smaller than the theoretical bound. Restricting the issue time range significantly reduces the search space for ILP solving. Treating the program dependence graph as an AOE network, we can compute earliest and latest issue times for each instruction.

Without considering machine resources, instructions can only be issued between earliest and latest times. However, cluster assignment may insert inter-cluster transfer instructions, and DSP issue width limitations may cause actual legal issue times to exceed the theoretical latest time. Therefore, we conservatively estimate instruction issue time intervals as follows:

a) For each basic block, traverse the program dependence graph without inter-cluster transfer instructions to compute earliest issue times:

where pred(i) denotes all direct and indirect predecessors of instruction i, and lat_{i,j} represents latency between instructions.

b) Traverse the dependence graph including transfer instructions and compute earliest issue times for each transfer instruction using the results from step (a).

c) Compute lower bounds for instruction issue times:

d) Traverse the program dependence graph without transfer instructions in reverse direction to compute latest issue times for instructions in basic block B:

where succ(i) denotes all direct and indirect successors of i. This latest time differs from the AOE network definition and is essentially the earliest time computed in the reverse direction.

e) Traverse the dependence graph including transfer instructions and compute latest issue times for each transfer instruction using results from step (d).

f) Compute upper bounds for instruction issue times:

For any basic block B, constraints limiting non-transfer instruction issue times to the estimated interval are:

For inter-cluster transfer instruction x ^{m_1,m_2}, the upper bound constraint is similar, but since the instruction may not be scheduled, its constraint is:

5.1 Handling Function Call Instructions

For simplicity, the ILP modeling above ignores function call instructions, a special case. The actual execution time of a call depends on the callee and cannot be determined precisely. When a function returns, return value registers are ready, so subsequent instructions using the return data need not wait for flow dependency latency, making previous dependency constraints inaccurate. Treating call instructions as basic block boundaries bypasses this issue but sacrifices some parallelism. An alternative approach makes call instructions occupy multiple clock cycles equal to maximum latency L, ensuring all dependent instructions can issue immediately after the call completes without modifying the model. Although this imprecisely calculates the last instruction issue time in the optimization objective, it does not affect the goal of approximating program execution time. Under this approach, no other instructions should occupy cycles t+1 through t+L when call instruction c issues at time t in basic block B:

5.2 Optimizing Solver Speed

Since DSP clusters are isomorphic, many symmetric cluster assignment solutions exist. Breaking these symmetries often accelerates ILP solving. The approach prefers assigning instructions to clusters with smaller numbers, implemented by adding:

6 Experimental Evaluation

We selected the HXDSP1042 as our evaluation platform—a typical clustered VLIW DSP developed by the 38th Research Institute of China Electronics Technology Group, featuring 16-issue width, 4 clusters, 13 pipeline stages, and inter-cluster communication via buses, widely used in radar signal processing. While ILP solving is a complex topic beyond this paper' s scope, we used the Gurobi solver to implement our algorithm. Test programs were selected from DSP-Stone, compiled with O2 optimization, replacing only the instruction cluster assignment and scheduling passes. We implemented several unified cluster assignment and scheduling algorithms as baselines, which operate on each basic block separately, while our method operates on entire functions.

Execution on the HXDSP1042 simulator shows our method achieves 20% to 35% performance improvement over LUCAS, the best-performing heuristic algorithm, across different benchmarks. Figure 9 [Figure 9: see original paper]

shows normalized execution times for several DSPStone benchmarks, using UAS as baseline 1. UAS-CC denotes UAS using completion time as heuristic, LSTC represents the algorithm from reference [9], and ILP denotes our method.

Although ILP is NP-hard, branch-and-bound often solves functions in general programs quickly. We randomly selected 4,000 functions from GSL, FANN, Zlib, and DSPStone, testing our algorithm on an Intel Core i5-8300H laptop. Results show approximately 81% of functions are solved within 0.5s, as shown in Table 2 .

7 Conclusion

This paper employs integer linear programming to implement instruction cluster assignment and scheduling for clustered VLIW DSPs. Experiments on HXDSP1042 demonstrate that our method effectively improves program performance with acceptable algorithm runtime. Although slower than heuristic algorithms, it achieves practical usability. One limitation is the coarse estimation of function execution time, which could be improved using high-level compiler information or deep learning methods for branch prediction. Another issue is the time overhead, stemming from using a large ILP model for all instructions, while many instructions' assignments and schedules are independent. Future work will explore partitioning functions into independent parts, modeling and solving each separately while guaranteeing global optimality, to further improve solving speed.

References

[1] Mattson P, Dally W J, Rixner S, et al. Communication scheduling [J]. ACM SIGOPS Operating Systems Review, 2000, 34(5): 82-92.
[2] Kessler R. The alpha 21264 microprocessor [J]. IEEE Micro, 1999, 19(2): 24-36.
[3] Ellis J R. Bulldog: a compiler for VLSI architectures [M]. Cambridge, MA: Mit Press, 1986.
[4] Lowney P G, Freudenberger S M, Karzes T J, et al. The multiflow trace scheduling compiler [M]// Instruction-Level Parallelism. Boston: Springer, 1993: 51-142.
[5] Desoli G. Instruction assignment for clustered VLIW DSP compilers: A new approach [M]. Palo Alto, California: Hewlett Packard Laboratories, 1998.
[6] Lapinskii V S, Jacome M F, De Veciana G A. Cluster assignment for high-performance embedded VLIW processors [J]. ACM Trans on Design Automation of Electronic Systems (TODAES), 2002, 7(3): 430-454.
[7] Ozer E, Banerjia S, Conte T M. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures [C]// Proc of the 31st Annual ACM/IEEE International Symposium on Microarchitecture. Piscataway, NJ: IEEE, 1998: 308-315.
[8] Kailas K, Ebcioglu K, Agrawala A. CARS: A new code generation framework

for clustered ILP processors [C]// Proc of the 7th International Symposium on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2001: 133-143.
[9] Zhang Xuemeng, Wu Hui, Xue Jingling. An efficient heuristic for instruction scheduling on clustered VLIW processors [C]// Proc of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems. 2011: 35-44.
[10] Porpodas V, Cintra M. LUCAS: latency-adaptive unified cluster assignment and instruction scheduling [J]. ACM SIGPLAN Notices, 2013, 48(5): 45-54.
[11] Fisher J A. Trace scheduling: A technique for global microcode compaction [J]. IEEE Trans on Computers, 1981, 30(07): 478-490.
[12] Wilken K, Liu J, Heffernan M. Optimal instruction scheduling using integer programming [J]. ACM SIGPLAN Notices, 2000, 35(5): 121-133.
[13] Kästner D, Winkel S. ILP-based Instruction Scheduling for IA-64 [J]. ACM SIGPLAN Notices, 2001, 36(8): 145-154.
[14] Mendis C, Amarasinghe S. goSLP: globally optimized superword level parallelism framework [J/OL]. Proc of the ACM on Programming Languages, 2018, 2(OOPSLA): 1-28. (2018-10-24) [2022-05-07]. https://doi.org/10.1145/3276480.
[15] Fisher J A. The optimization of horizontal microcode within and beyond basic blocks: An application of processor scheduling with resources [D]. New York: New York University, 1979.
[16] Wang Yulin, Zheng Qilong. Instruction scheduling optimization for clustered VLIW HXDSP [J]. Microcomputer & Its Applications, 2017(11): 23-26.
[17] Zalamea J, Llosa J, Ayguadé E, et al. Modulo scheduling with integrated register spilling for clustered VLIW architectures [C]// Proc of the 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34. Piscataway, NJ: IEEE, 2001: 160-169.
[18] Aleta A, Codina J M, Sánchez J, et al. AGAMOS: A graph-based approach to modulo scheduling for clustered microarchitectures [J]. IEEE Trans on Computers, 2009, 58(6): 770-783.
[19] Mahlke S A, Lin D C, Chen W Y, et al. Effective compiler support for predicated execution using the hyperblock [J]. ACM SIGMICRO Newsletter, 1992, 23(1-2): 45-54.
[20] Llosa J, González A, Ayguadé E, et al. Swing module scheduling: a lifetime-sensitive approach [C]// Proc of the 22nd Conference on Parallel Architectures and Compilation Technique. Piscataway, NJ: IEEE, 1996: 80-86.
[21] Ferrante J, Ottenstein K J, Warren J D. The program dependence graph and its use in optimization [J]. ACM Trans on Programming Languages and Systems (TOPLAS), 1987, 9(3): 319-349.
[22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual [EB/OL]. [2022-05-07]. https://www.gurobi.com.
[23] Institute for Communication Technologies and Embedded Systems. DSP-Stone [EB/OL]. [2022-05-07]. https://www.ice.rwth-aachen.de/research/tools-projects/closed-projects/dspstone.

[24] Gough B. GNU scientific library reference manual [M]. [S.l.]: Network Theory Ltd., 2009.
[25] Nissen S. Implementation of a fast artificial neural network library (fann) [EB/OL]. (2003) [2022-05-07]. http://fann.sourceforge.net/report.
[26] Bieber D, Sutton C, Larochelle H, et al. Learning to execute programs with instruction pointer attention graph neural networks [C/OL]// Advances in Neural Information Processing Systems. New York, NY: Curran Associates, Inc., 2020: 8626-8637. https://proceedings.neurips.cc/paper/2020/file/62326dc7c4f7b849d6f013ba46489d6c-Paper.pdf.

Note: Figure translations are in progress. See original paper for figures.