Integer Linear Programming vs. Graph-Based Methods in Code Generation

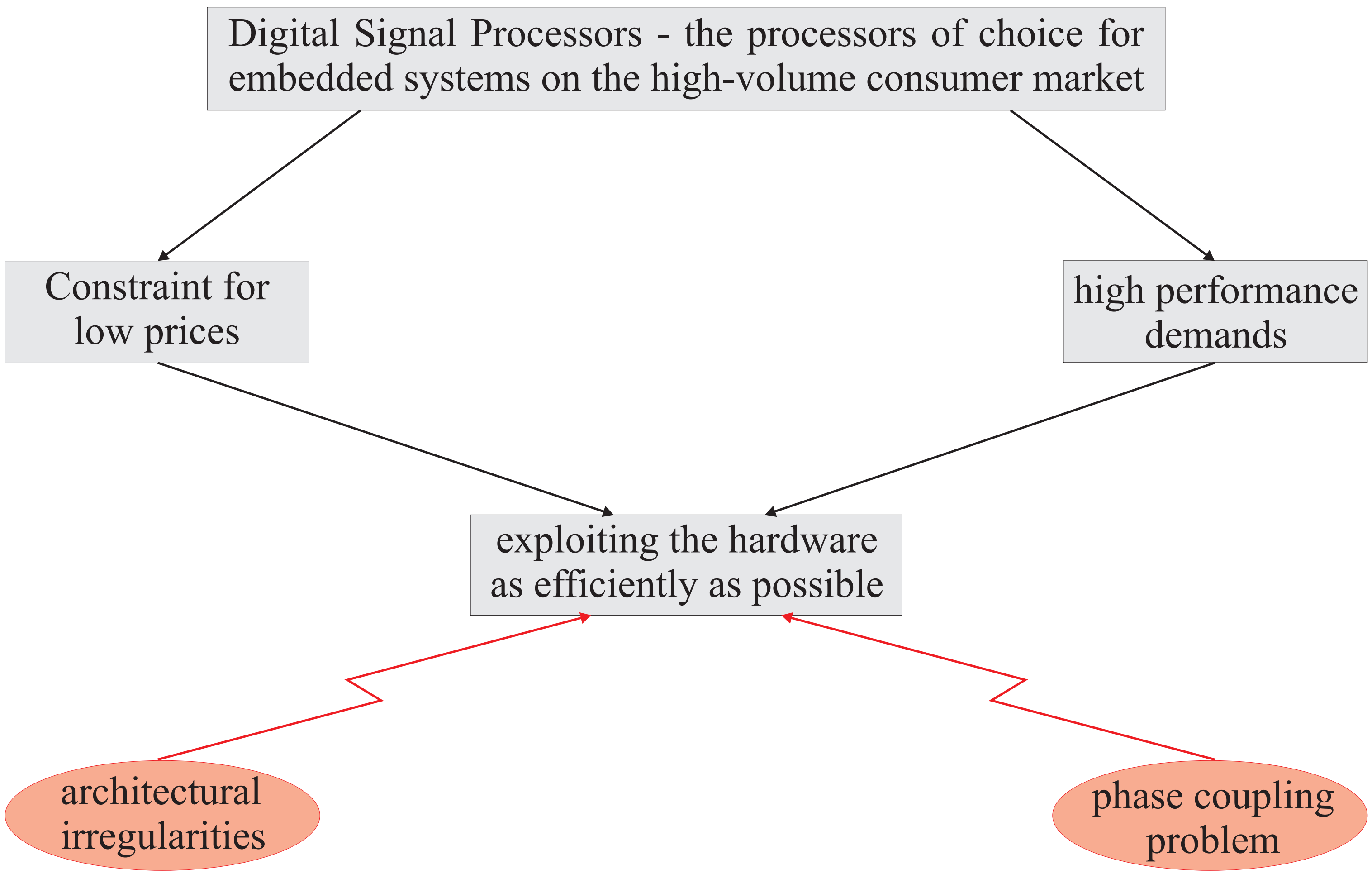
Daniel Kästner   Marc Langenbach

{kaestner,mlangen}@cs.uni-sb.de

Germany

- Code generation for DSPs
- Conventional graph-based algorithms
- Basic concepts of integer linear programming (ILP)
- Description of one of the investigated ILP formulations: SILP (Zhang, 1996)
- Architecture of the ADSP-2106x Sharc
- Extensions to the ILP formulations
- ILP-based approximations
- Implementation and results
- Conclusion

The Problem of Code Generation



Overview

Code Generation

Graph-based

ILP

SILP

Architecture

Extensions

Approximations

Results

Conclusion

Instruction Scheduling and Register Allocation

Instruction Scheduling → ← Register Allocation

Registers are used to exploit instruction level parallelism

Registers are used to minimize the number of memory references

↓

Integer Linear Programming:

- Homogeneous problem description
- Calculation of an optimal solution
- Calculation of lower bounds on the optimal execution time of the input program.

Overview
Code Generation
Graph-based
ILP
SILP
Architecture
Extensions
Approximations
Results
Conclusion

Interdependence

$R1 = R1 * R4$
$R2 = R8 + R12$

$\Rightarrow$ $R1 = R1 * R4, \; R2 = R8 + R12$

$R1 = R1 * R3$
$R2 = R8 + R12$

$\Rightarrow$

$R1 = R1 * R3$

$R2 = R8 + R12$

List Scheduling

Input: - set of microoperations
- data dependence graph (DDG)

begin
$IList = \emptyset$
repeat
choose a microoperation m which satisfies three conditions:
1) it is ready, i.e. all predecessors in the DDG have already been scheduled
2) its priority is the highest one among the priorities of all ready instructions (heuristics: first fit, ...)
3) the operation can be executed in parallel to all other operations contained in the last instruction

insert m into the last instruction of $IList$
when there is no ready operation that can be inserted into the last instruction, a new instruction is added to the list
until all operations have been scheduled
end

Global Scheduling Algorithms

- Trace scheduling (Fisher, 1981)
- Percolation scheduling (Nicolau, 1985)
- Region scheduling (Gupta/Soffa, 1990)

→ Impact of heuristics is very important

Register Allocation

- Graph-coloring techniques: find a k-coloring in the register interference graph

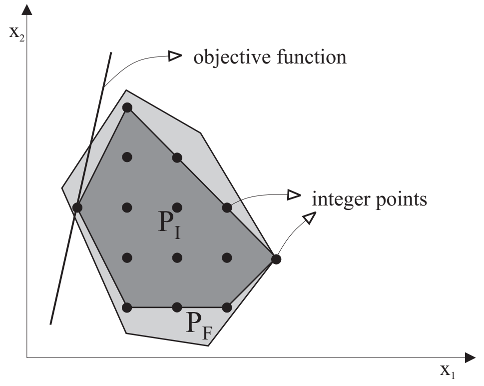
→ Heuristics are used, problem of instruction scheduling is ignored

Integrated Approach

- Bradlee (1991):

Basics of Integer Linear Programming

- With ILP, integrating instruction scheduling and some aspects of register allocation is possible in one unique problem formulation
- ILP is $\mathcal{NP}$-hard $\Rightarrow$ need for well-structured formulation

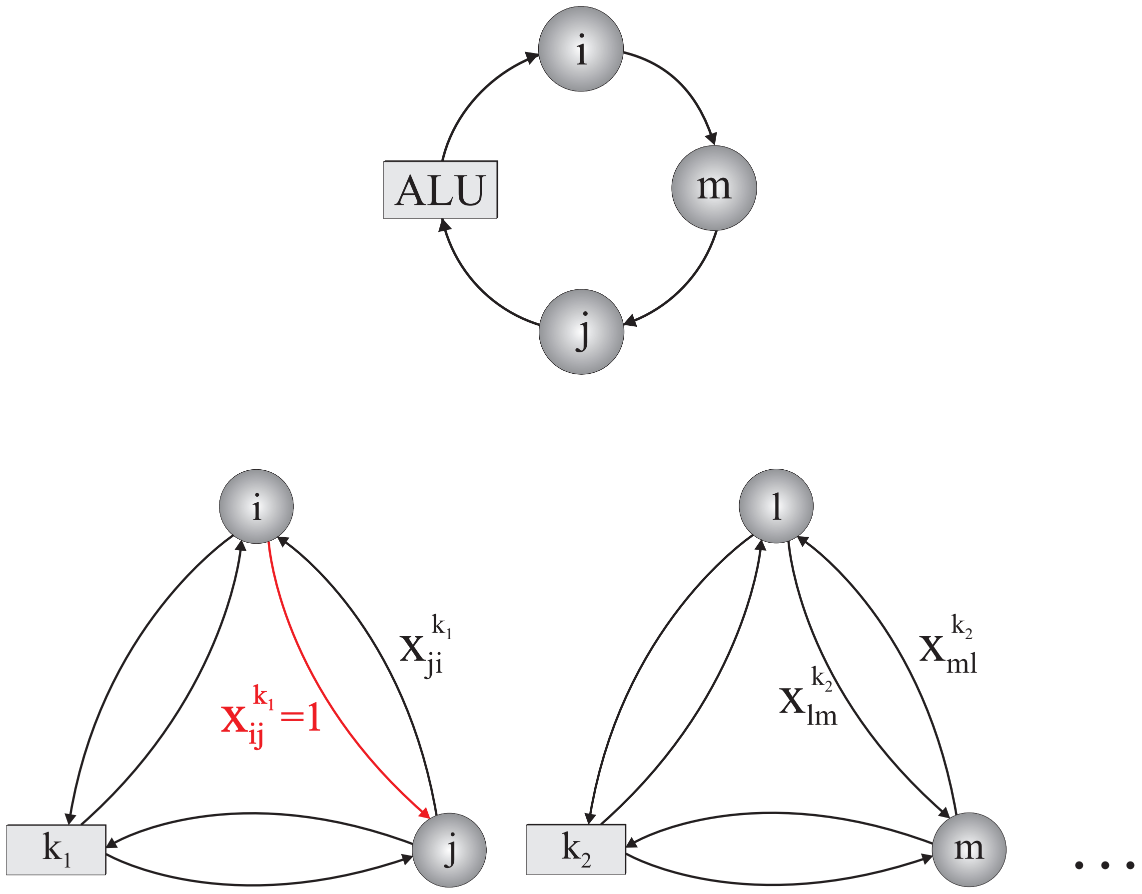


- $P_F = P_I \Rightarrow$ integer optimal solution can be calculated by linear programming algorithms in polynomial time
- P_F must be as close as possible to P_I; the sharper this approximation is, the faster can the ILP be solved

Investigated ILP Formulations

- OASIC (Optimal Architectural Synthesis with Interface Constraints, Gebotys/Elmasry 1992/93)
- SILP (Scheduling and Allocation with Integer Linear Programming, Zhang 1996)

Constraints



Resource Flow Graph

Overview
Code Generation
Graph-based
ILP
SILP
Architecture
Extensions
Approximations
Results
Conclusion

SILP (Zhang, 1996)

- t_i starting time for the execution of operation i
- $x^k_{ij} = 1 \approx$ operation i passes an instance of resource type k to operation j
- Objective function

$$\min M_{steps}$$

subject to

- Time constraints
- Precedence constraints
- Flow conservation constraints
- Assignment constraints
- Resource constraints
- Serial constraints

$$x^k_{jn} \in \{0, 1\}$$

Constraints

$$\min M_{steps}$$

<u>Time Constraints</u>

$$t_j \leq M_{steps} \qquad \forall j$$

t_j: Starting time of operation j

Constraints

Precedence Constraints

$$t_j - t_i \geq w_i$$

$\forall$ edges (i, j) in the data dependence graph

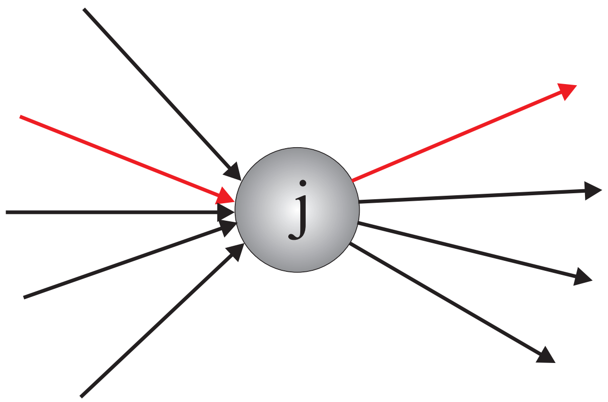
w_i: Execution time of operation i

Overview
Code Generation
Graph-based
ILP
SILP
Architecture
Extensions
Approximations
Results
Conclusion

Constraints

Flow Conservation Constraints

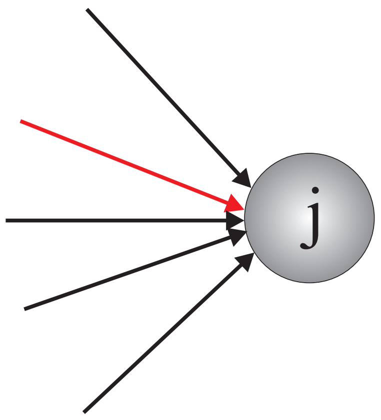
$$\sum_{(i,j)\in E_{F^k}} x_{ij}^k - \sum_{(j,m)\in E_{F^k}} x_{jm}^k = 0 \qquad \forall \ operations\ j$$



Constraints

Assignment Constraints

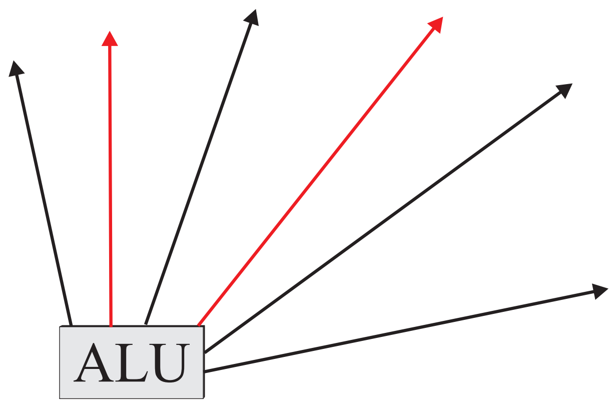
$$\sum_{\substack{k \in V_k: \\ j\ can\ be\ executed\ by\ k}} \sum_{(i,j) \in E_{F^k}} x_{ij}^k = 1 \qquad \forall j \in V_D$$



Constraints

Resource Constraints

$$\sum_{(k,j) \in E_{F^k}} x_{kj}^k \leq R_a \qquad \forall k \in V_k$$



Constraints

Serial Constraints

$$t_j - t_i \geq z_i + \left(\sum_{\substack{k \in V_k: \\ (i,j) \in E_{F^k}}} x_{ij}^k - 1 \right) \alpha_{ij} \qquad \forall (i,j) \in E_{F^k}$$

$$\begin{tikzcd} i \arrow[r, bend right, "x_{ij}^{k}=1"'] & j \end{tikzcd} \quad \Rightarrow \quad t_j - t_i \geq z_i$$

α_{ij}: Large constant

z_i: Latency

Overview
Code Generation
Graph-based
ILP
SILP
Architecture
Extensions
Approximations
Results
Conclusion

Integration of Register Assignment

- Register assignment can be integrated by additional constraints; these constraints model edges in the DDG which represent lifetimes of a value
- Register assignment $\approx$ determine the physical register that is used to store a value which has been selected to reside in a register by the allocator
- Register allocation $\rightarrow$ insertion of spill code
- ILPs are created for a fixed set of instructions (in OASIC and SILP)
  $\Rightarrow$ complete register allocation cannot be integrated into a unique ILP formulation

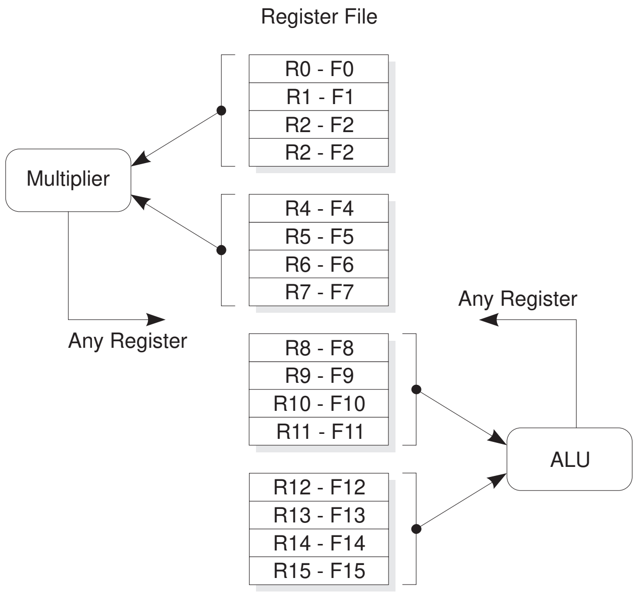
Overview
Code Generation
Graph-based
ILP
SILP
Architecture
Extensions
Approximations
Results
Conclusion

ADSP-2106x Sharc

- 32-bit digital signal processor
- Three functional units: ALU, multiplier, shifter
- Multifunctional instructions: ALU- and multiplier-operations can be executed in parallel
- Register file
  - Two sets of sixteen 40-bit registers
  - Fixed and floating point data
- On-chip data memory DM
- On-chip program memory PM, both instruction and data
- Accesses to PM and DM can be performed in parallel

ADSP-2106x Sharc

- Parallel execution of ALU and multiplier is only possible for a subset of all ALU- and multiplier-operations
- Operands have to reside in special register groups



- Arithmetic operations can be performed in parallel with a DM- and a PM-access but, e.g., control operations not

Extensions to the ILP Formulations

- Forbidding parallel execution of operations i and j if this is not prevented by data or control dependences but incorrect for our architecture
- Multifunctional operations: when ALU-operations and multiplier-operations are executed in parallel then the corresponding definitions must write to the correct register groups

$\Rightarrow$ Formulated with binary variables

$\Rightarrow$ Complexity increases

Overview
Code Generation
Graph-based
ILP
SILP
Architecture
Extensions
Approximations
Results
Conclusion

Global Optimizations

- SILP considers only sequential code
- OASIC suggests to consider all valid execution paths in the program separately
- Method to incorporate the control structure of the program into an ILP
  - For each instruction j, all basic blocks are determined which underly the same control conditions as the original basic block containing j: $B_1, B_2, \ldots, B_k$
  - The disjunction $j \in B_1 \vee j \in B_2 \vee \cdots \vee j \in B_k$ is formulated by linear constraints and added to the ILP formulation

Overview

Code Generation

Graph-based

ILP

SILP

Architecture

Extensions

Approximations

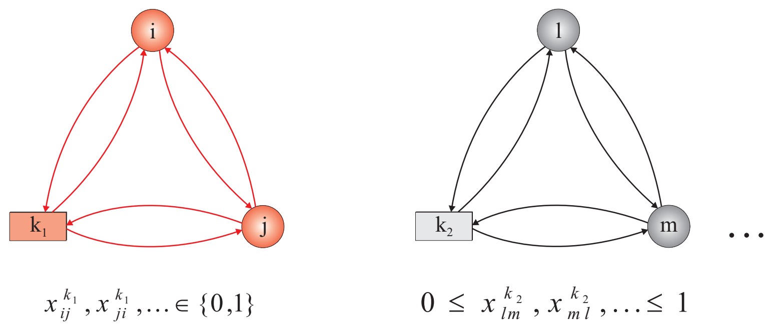
Results

Conclusion

- Approximations during the solution process of the ILPs
- Use the complete program information but instead of an exact solution consider heuristically determined partial relaxations
- Relaxation: allow for some variables to take non-integral values $\rightarrow$ computation time is significantly reduced
- approximation by rounding for the SILP approach
  - Replace $x_{ij}^k \in \{0, 1\}$ by $0 \leq x_{ij}^k \leq 1$ and solve this relaxation
  - Fix the variable with a non-integral value closest to 0 or 1 to that integer and solve this MILP
  - Repeat this step until an integral solution is found

  $\rightarrow$ Solution quality not convincing

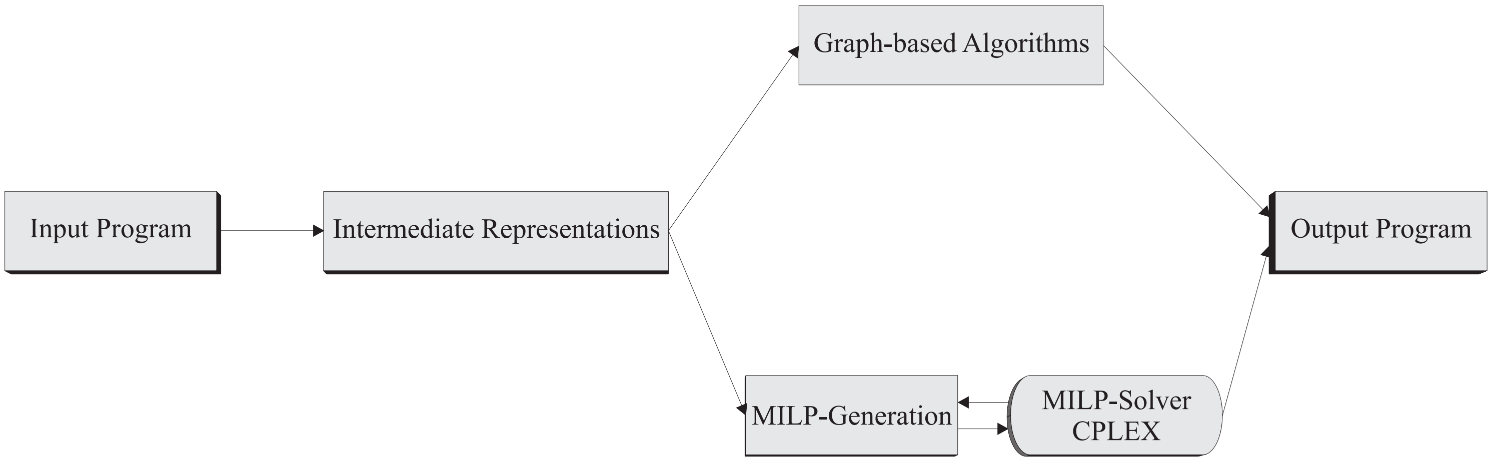
- Rounding approach turned out not to be promising
- More efficient approximations, e.g. isolated flows (IF), stepwise approximation (SA), SAIF
- Isolated flows:

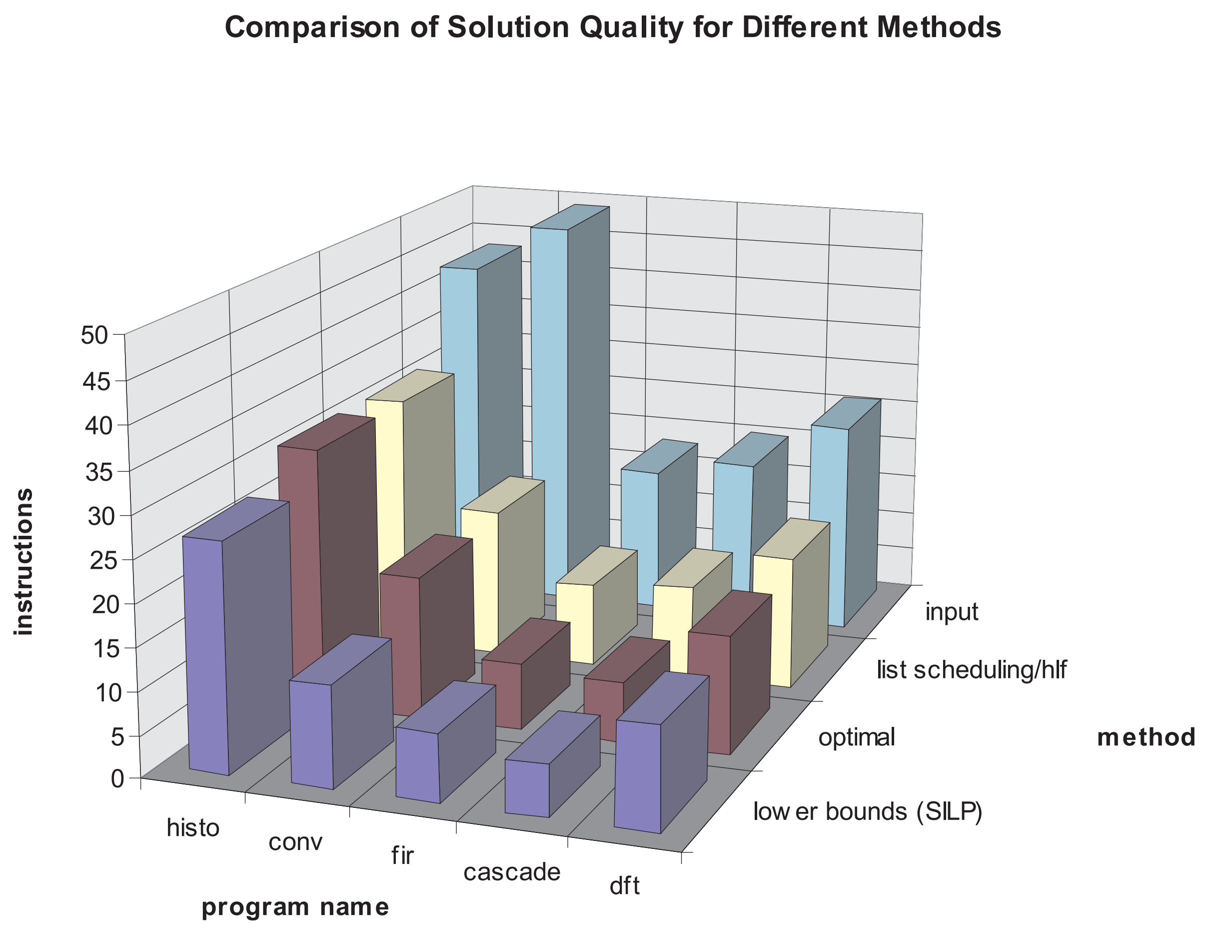


$\Rightarrow$ Computation time is reduced; yet mostly optimal solutions are produced

- SAIF: considerable acceleration with mostly optimal solutions; e.g. a problem whose exact solution took more than 24 hours could be solved in less than 2 minutes with optimal results

Implementation





Comparison of Solution Quality for Different Methods
instructions
50
45
40
35
30
25
20
15
10
5
0
histo
conv
fir
cascade
dft
program name
input
list scheduling/hlf
optimal
low er bounds (SILP)
method

Comparison of Solution Quality

program	lower bounds	optimal	list scheduling
fir	8	8	10
cascade	6	7	9
dft	12	14	16
whetp3	16	20	21
histo	27	31	31
conv	12	17	18
deviation	-15%	0%	+13%

Speedup by Approximations for SILP

name	mode	method	instr	CPU-time
fir	is+ra	exact	8	1.69 sec
fir	is+ra	SAIF	8	19.58 sec
cascade	is	dexact	7	0.19 sec
cascade	is	SAIF	7	0.27 sec
cascade	is+ra	exact	-	> 24 h
cascade	is+ra	SAIF	7	86.72 sec
dft	is	exact	14	47.42 sec
dft	is	SAIF	14	42.58 sec
dft	is+ra	exact	-	> 24 h
dft	is+ra	SAIF	14	9 min 20 sec
whetp3	is	exact	20	2h 4 min
whetp3	is	SAIF	21	85.96 sec
histo	is	exact	-	> 24 h
histo	is	SAIF	31	1 h 2 min
conv	is	exact	17	2 h 1 min
conv	is	SAIF	17	53.66 sec

program	lower bound	CPU time	deviation
cascade	6	6.76 sec	14%
dft	12	24.142 sec	14%
histo	27	156.45 sec	12.9%

- With ILP, instruction scheduling and register assignment can be integrated in an unique problem formulation, adapted to an irregular target architecture, suited to global optimizations
- Calculation time for exact solutions of the ILPs is too high for large programs
- Approximations based on ILP provide mostly optimal solutions in acceptable time for the considered programs
- Lower bounds on the optimal instruction number can be calculated in short time by partial relaxations of the ILPs
  $\Rightarrow$ estimations of the quality of an approximative solution is possible even without an optimal solution being available

- Conventional algorithms are very fast
- The best conventional algorithm exceeds the optimal instruction number by 13% on average $\approx$ same quality as for the ILP-based lower bounds

$\Rightarrow$ Instruction scheduling and register allocation only based on ILP is not practicable for reasons of complexity. An embedding of ILP methods in a conventional algorithm can be considered promising where an ILP-optimization is performed for suitable code sequences.