

Optimal vs. Heuristic Integrated Code Generation for Clustered VLIW Architectures

Mattias Eriksson Oskar Skoog Christoph Kessler

Programming Environments Laboratory
Dept. of Computer and Information Science
Linköping University, Sweden

SCOPES, 14th March 2008



Linköping University

Outline

Introduction

Integrated Code Generation

Optimal Method — Integer Linear Programming

Heuristic Method — Genetic Algorithm

Evaluation

Experimental Results

Conclusions

Our Contributions

Outline

Introduction

Integrated Code Generation

Optimal Method — Integer Linear Programming

Heuristic Method — Genetic Algorithm

Evaluation

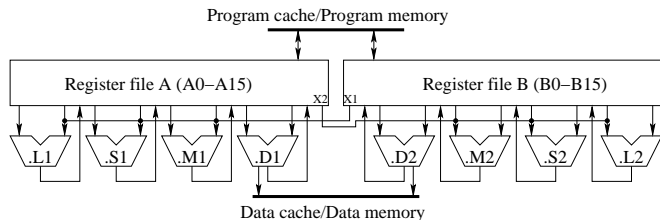
Experimental Results

Conclusions

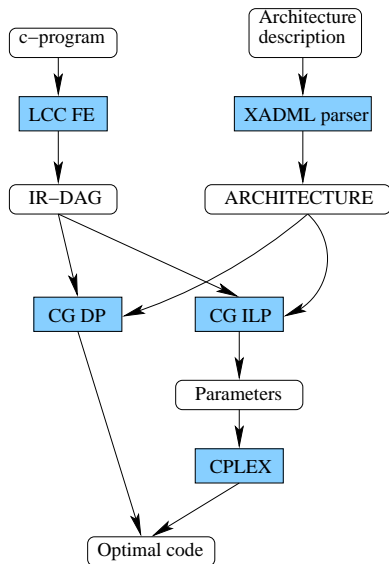
Our Contributions

Integrated Code Generation

- ▶ The three phases of code generation:
 - ▶ Instruction selection,
 - ▶ Instruction scheduling and
 - ▶ Register allocation
- ▶ They are normally done in some sequence.
- ▶ But making decisions in all phases simultaneously gives more opportunity for optimization.
- ▶ Especially for clustered architectures. Where a value is stored has great impact on how it may be used.



The OPTIMIST Framework



- ▶ The framework that we work in is called OPTIMIST
- ▶ Currently OPTIMIST has two methods for optimal integrated code generation (CG)
 - ▶ Dynamic programming
 - ▶ Integer Linear Programming
- ▶ We extend OPTIMIST by:
 - ▶ Generalizing ILP to clustered architectures
 - ▶ Adding a new heuristic CG based on genetic algorithms

Outline

Introduction

Integrated Code Generation

Optimal Method — Integer Linear Programming

Heuristic Method — Genetic Algorithm

Evaluation

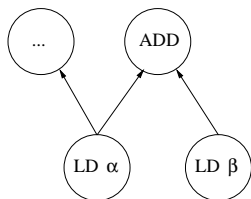
Experimental Results

Conclusions

Our Contributions

Integer Linear Programming

Formulating ILP for clustered architectures is complicated.



- ▶ **Non-clustered:** Both loads must precede the ADD by latency of LD time units. Modeled by simple precedence constraints.
- ▶ **Clustered:** If α is loaded into reg. file A and β is loaded into reg. file B
 - ▶ One of the loaded values may have to be copied
 - ▶ \rightarrow it must precede ADD by more than latency of LD.
- ▶ α may be alive in more than one register file \rightarrow live ranges are more complicated.

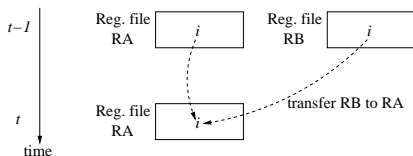
Integer Linear Programming

- ▶ We have the binary variable

$$r_{rf,i,t}$$

which is 1 iff IR-node i is available in register file rf at time slot t

- ▶ Hence, $r_{rf,i,t}$ may be 1 [$r_{rf,i,t} \leq \dots$] when
 - ▶ The value from i was available in file rf at time $t - 1$, or
 - ▶ The value i was transferred to rf from another register file at time t , or
 - ▶ An instruction just finished calculating i and stored the resulting value in rf at time t



- ▶ We model data-flow dependencies by constraints on r .
 $[r_{rf,i,t} \geq \dots]$

Outline

Introduction

Integrated Code Generation

Optimal Method — Integer Linear Programming

Heuristic Method — Genetic Algorithm

Evaluation

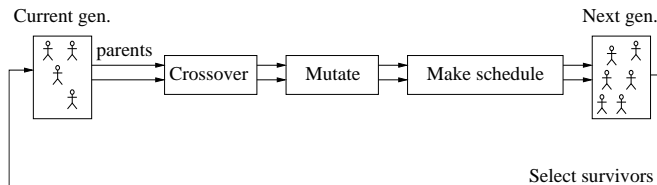
Experimental Results

Conclusions

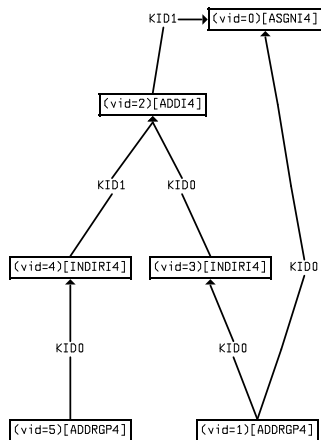
Our Contributions

Genetic Algorithms

- ▶ The ILP gives optimal solutions but can be time consuming for large input → we need a heuristic.
- ▶ Genetic algorithms may be used to solve optimization problems with huge solution spaces.
 - ▶ Idea: mimic natural selection (evolution)
 - ▶ Strong individuals survive and spread their genes.



The Genes



- ▶ This IR-DAG represents the basic block $\{a = a + b\}$
- ▶ A valid schedule:

```
LDW.D1 _a, A15 || LDW.D2 _b, B15
NOP ; Latency of load is 4
NOP
NOP
NOP
ADD.D1X A15, B15, A15
MV.L1 _a, A15
```

- ▶ Extract genes:
 - ▶ node-order={1,5,3,4,2,0}
 - ▶ instr-map={3 → “LDW.D1”,
4 → “LDW.D2”,
2 → “ADD.D1X”,
0 → “MV.L1”}

Crossover and Mutation

- ▶ **Crossover:** Find a *matching point* and swap genes
- ▶ At a matching point the parents' partial schedules have
 - ▶ scheduled the same IR-nodes
 - ▶ compatible resource usage
 - ▶ the same pending latencies

Example crossover:

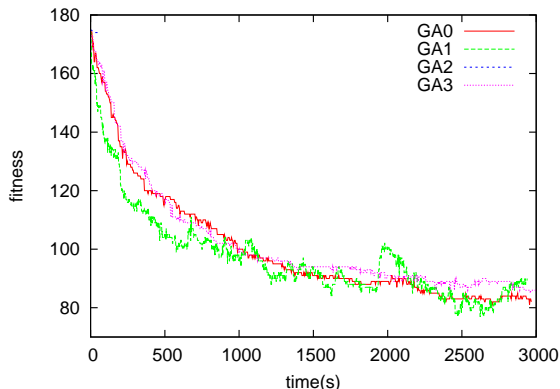
node-order-1={1,2,3,4}, node-order-2={2,1,4,3}

If the partial schedules for nodes 1 and 2 are compatible, we have a matching point!

- ▶ **Mutation** comes 2 flavors.
 - ▶ Change positions of nodes in the node-order
 - ▶ Change instruction for a node
- ▶ Mutation may create an invalid individual. Sometimes this can be repaired, other times the individual must be discarded.

Convergence Behavior

	mutation	nr. of ind.	parents
GA0	50%	50	die
GA1	75%	10	survive
GA2	25%	100	die
GA3	50%	50	survive



- The diagram shows example convergence for a DAG with 138 nodes.

Outline

Introduction

Integrated Code Generation

Optimal Method — Integer Linear Programming

Heuristic Method — Genetic Algorithm

Evaluation

Experimental Results

Conclusions

Our Contributions

Summary of Evaluation

- ▶ Our benchmarks are *all basic blocks of relevant size* in **mpeg2** and **jpeg** from Mediabench.
- ▶ The target architecture is TI-C62x.
 - ▶ 2 clusters
 - ▶ can issue 8 instructions / clock cycle
- ▶ Time limit for ILP is 900 seconds.
- ▶ Time for code generation is 650-800 seconds.
- ▶ GA runs one thread per individual (on 2 cores), ILP uses only one thread.

	ILP	GA0	GA1
Finds solution	33	81	81
Solution known to be optimal	33	28	25
Finds solution when ILP fails	-	48	48
Finds solution better than GA0	5	-	9
Finds solution better than GA1	8	14	-

Outline

Introduction

Integrated Code Generation

Optimal Method — Integer Linear Programming

Heuristic Method — Genetic Algorithm

Evaluation

Experimental Results

Conclusions

Our Contributions

Our Contributions

- ▶ We have formulated an ILP for optimal integrated CG for clustered VLIW architectures.
- ▶ We have also created a heuristic method for integrated CG based on genetic algorithms.
- ▶ We have compared the ILP to GA and found that:
 - ▶ ILP finds optimal solutions for moderately large problems. And the rest of the problems may be solved by the GA.
 - ▶ When optimal solution is known, GA finds it 85% of the time