IR-Level vs. Machine-Level If-Conversion for Predicated Architectures [1]

Alexander Jordan, Nikolai Kim, Andreas Krall
{ajordan,kim,andi}@complang.tuwien.ac.at

FAKULTÄT FÜR !INFORMATIK

Institute of Computer Languages
University of Technology Vienna

ODES-10: Optimizations for DSP and Embedded Systems

[1] This work is supported by the Austrian Science Fund (FWF) under contract P21842, *Optimal Code Generation for Explicitly Parallel Processors*.

Outline

Implementation

- Two iterative If-Conversion variants
- Different levels of code abstraction
- Tail duplication on the machine level
- Implemented within LLVM 2.9
- Targeting TI's TMS320C64X DSP

Evaluation

- Multi-Issue-Multi-Cluster compiler setup
- Profile information vs. profile estimation
- Based on a cycle accurate simulator
- MiBench, DSPStone, Olden, BenchmarkGames tests

If-Conversion: Introduction

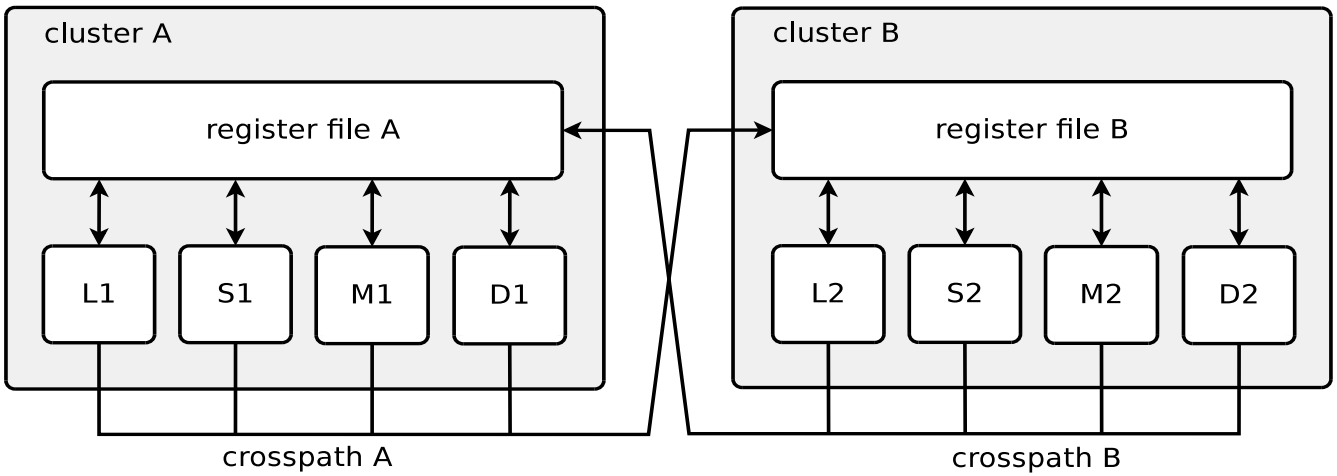
Basics

- Requires hardware support
- Removes conditional branches
- Naturally enlarges scheduling scope
- Possible application on different levels
- Increases register pressure

Idea

- Turn control dependences into data dependences
- Favor frequent, penalize less frequent regions
- Eliminate conditional branches, merge basic blocks

Target architecture



Texas Instruments TMS320C64X

- Clustered VLIW architecture, 2 clusters
- 4 functional units, 32 GP registers per cluster
- 3 predicate registers per cluster
- SIMD subset, full predication support

Basic application

Example

```
void foo (int z, int val) {
  int x = val;
  if (z > 10) x = x + 1;
  return x * x;
}
```

```
foo:
   move val, t1
   cmp_leq z, 10, t2
   branch_eq t2, 1, tail

body:
   add t1, 1, t1

tail:
   mul t1, t1, t1
   ret t1
```

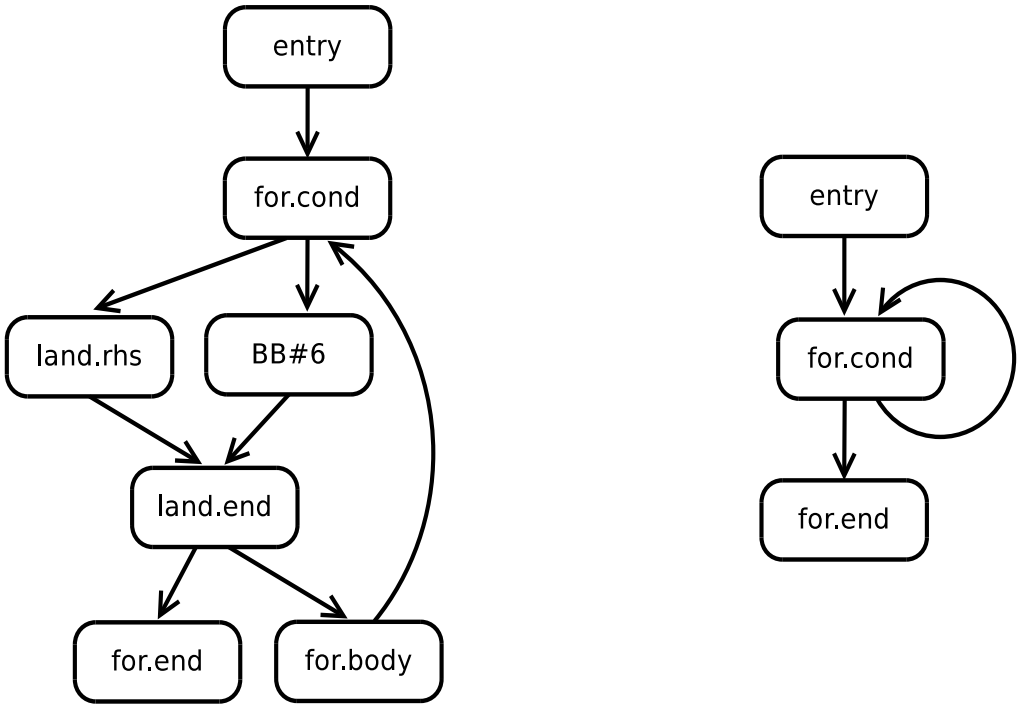
```
foo:
        move val, t1
        cmp_leq z, 10, p
  [!p] add t1, 1, t1
        mul t1, t1, t1
        ret t1
```

Iterative If-Conversion

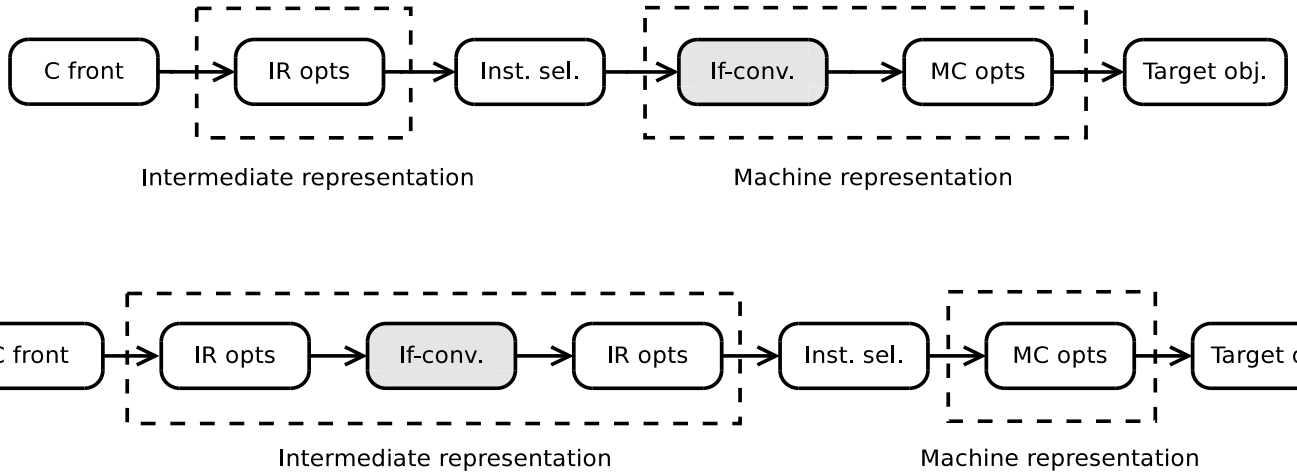
Process bottom-up, repeat until a threshold is reached.

Repeat steps

- Identify basic convertible patterns
- Fold predicates and convert identified patterns



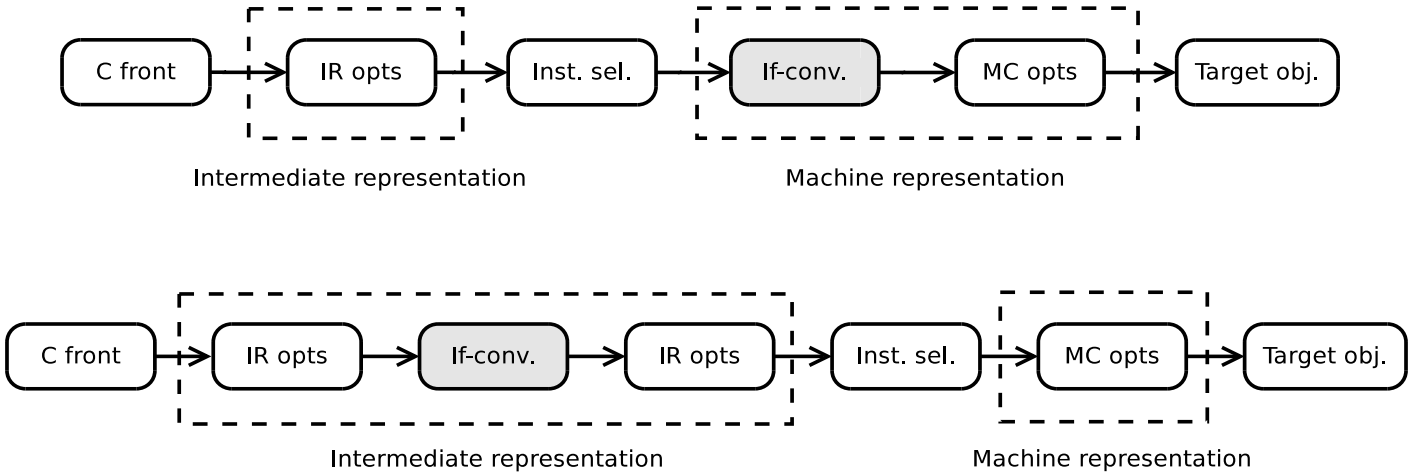
Code abstraction: machine level (ML)



Features

- Straightforward implementation
- Exact target specific information (instructions, latencies)
- Precedes register allocation, manipulates virt. registers
- Utilizes extra passes to generate precise profile information
- Integrates a tail duplication mechanism

Code abstraction: intermediate representation (IR)



Features

- Early SSA-transformation, implementation more involved
- Target information not present, thus modeled inexactly
- Instrumentation and profile loading provided by LLVM
- Can be combined with LLVM-IR optimizations

Conversion profitability

Estimation details

- Statically adjustable for different optimization levels
- Conservative register pressure estimation
- Geared toward branch elimination, no knowledge about scheduling

C: candidate region containing n basic blocks
cycles, weight: execution time, frequency for a basic block

$$Cost(C) = branch\_cost(C) + \sum_{i=1}^{n} cycles(Block_i) * weight_i$$

$$Cost_{conv}(C) = cycles(\bigcup_{i=1}^{n} Block_i) * weight_C$$

Performance evaluation: statistic

Benchmark	Baseline (kCycles)	IR-ifconv prof. (kCycles)	ML-ifconv prof. (kCycles)	ML-ifconv estim. (kCycles)	IR-speedup prof. (%)	ML-speedup prof. (%)	ML-speedup estim. (%)
Simple-bubblesort	1.70	1.38	1.55	1.55	18.46	8.37	8.37
Simple-quicksort	1.41	1.17	1.26	1.26	16.83	11.01	10.44
DSP-fir2dim	6.20	5.50	5.66	5.33	11.26	8.76	13.92
DSP-lms	1.22	1.16	0.98	0.98	5.16	19.98	19.98
DSP-fft	90.07	89.42	89.07	89.41	0.72	1.11	0.73
DSP-matrix1	32.84	26.19	24.62	24.62	20.26	25.02	25.02
DSP-matrix2	29.28	22.79	24.09	23.90	22.18	17.72	18.38
DSP-n_complex_updates	1.42	1.42	1.30	1.30	0.00	8.64	8.64
DSP-n_real_updates	1.05	0.69	0.92	0.92	34.09	12.63	12.63
DSP-startup	7.26	5.44	5.68	5.68	25.08	21.80	21.80
MB-adpcm	1136.39	526.81	599.57	599.57	53.64	47.24	47.24
MB-basicmath	1919.05	1757.54	1919.05	1919.05	8.42	0.00	0.00
MB-bitcount	1002.55	820.11	810.59	810.59	18.20	19.15	19.15
MB-blowfish	393.38	395.25	370.44	370.44	-0.48	5.83	5.83
MB-CRC32	7376.18	7376.18	5310.91	5310.91	0.00	28.00	28.00
MB-dijkstra	87441.65	71757.59	79975.22	82901.63	17.94	8.54	5.19
MB-FFT	49896.15	50350.01	49910.29	49910.29	-0.91	-0.03	-0.03
MB-qsort	57888.96	56059.27	57788.96	57788.96	3.16	0.17	0.17
MB-stringsearch	5312.74	5304.64	4568.07	5246.85	0.15	14.02	1.24
MB-susan	51545.54	45264.09	43007.53	43007.53	12.19	16.56	16.56
Olden-bisort	48526.77	46456.66	45792.90	45792.90	4.27	5.63	5.63
BG-fannkuch	238.99	234.09	220.21	214.24	2.05	7.86	10.35
BG-nsieve-bits	32078.41	26262.03	24194.72	24194.72	18.13	24.58	20.71
Average					**13.82**	**14.38**	**13.85**

Figure: Detailed comparison statistic

Performance evaluation: ML vs. IR

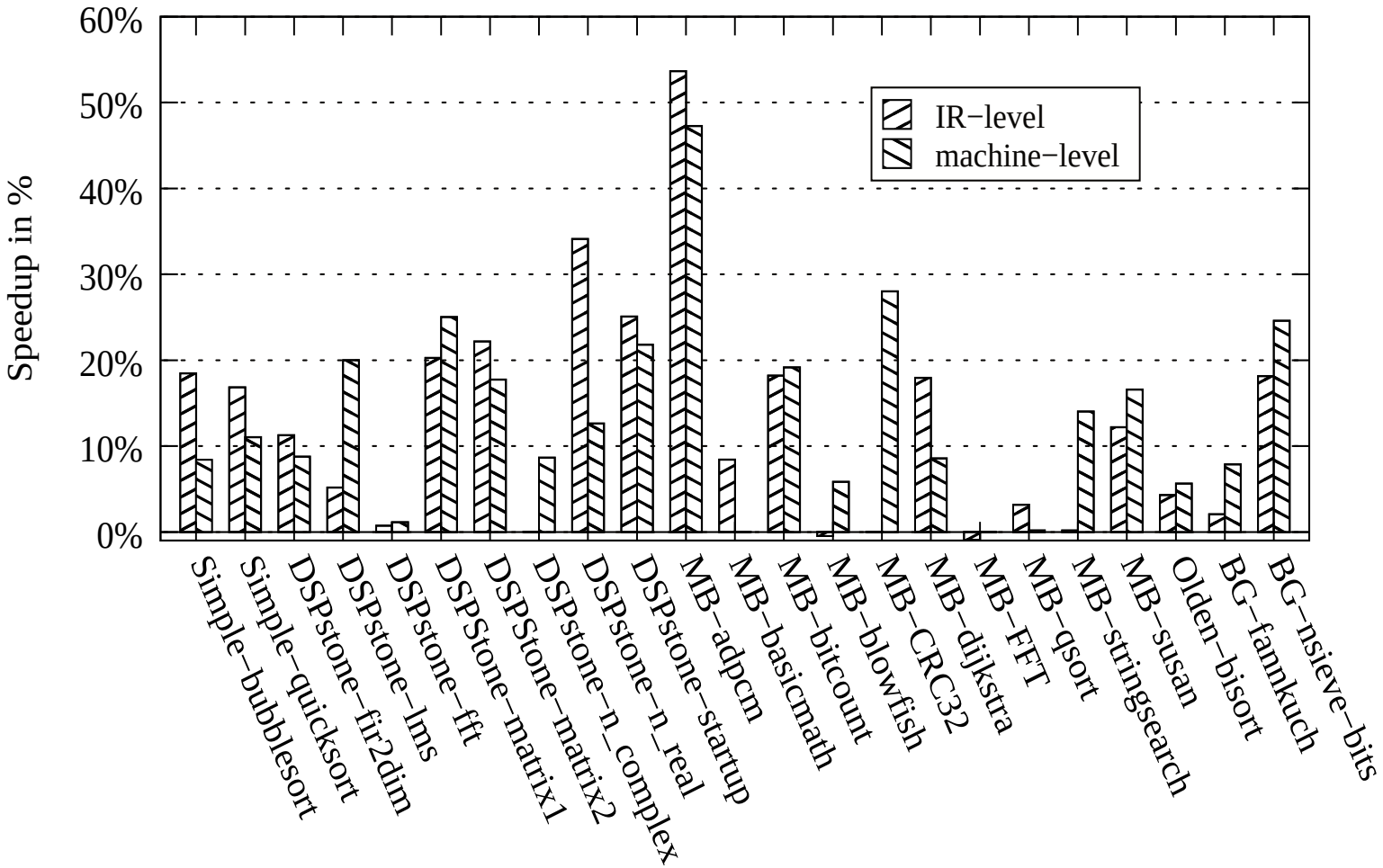


Figure: If-Conversion: ML vs. IR

Conclusions

Machine level If-Conversion...

- performs slightly better when compared to the IR-variant
- more traditional, easy to implement and to adapt
- performs equally well in absence of profile information
- exposes a more predictable optimization behavior
- is less flexible due to the machine pass pipeline

LLVM-IR If-Conversion...

- also achieves a substantial speedup
- requires more implementational effort
- preserves SSA-properties of the code
- is combinable with existing SSA-transformations
- is therefore also subject to phase ordering issues

Thank You

Thank you for being my audience!