Integrated Modulo Scheduling and Cluster Assignment for TMS320C64x+ Architecture [1]

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

Institute of Computer Languages
University of Technology Vienna

ODES-11: Optimizations for DSP and Embedded Systems

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

Outline

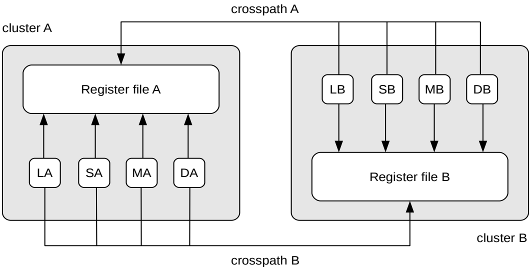
Implementation

- Swing modulo scheduling extension/adaptation
- Two different cluster assignment heuristics
- Implemented within LLVM 2.9
- Targeting TI's TMS320C64X DSP

Evaluation

- Taking UAS, ILP as baseline
- Based on a cycle accurate simulator
- MiBench, mediabench, DSPStone, BenchmarkGames, SingleUnit tests
- 35 kernels in total, 14 most representative presented

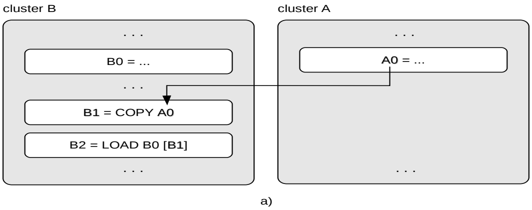
Target architecture



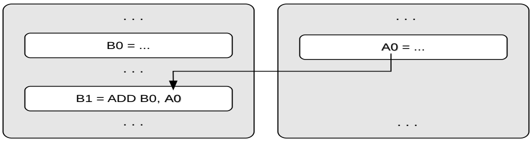
Texas Instruments TMS320C64X

- Clustered VLIW architecture, 2 clusters
- 4 functional units, 32 GP registers per cluster
- 3 predicate registers per cluster, 6 cycles branch latency
- DSP, SIMD subset, predication, soft. pipelining buffer

Intercluster communication



a)

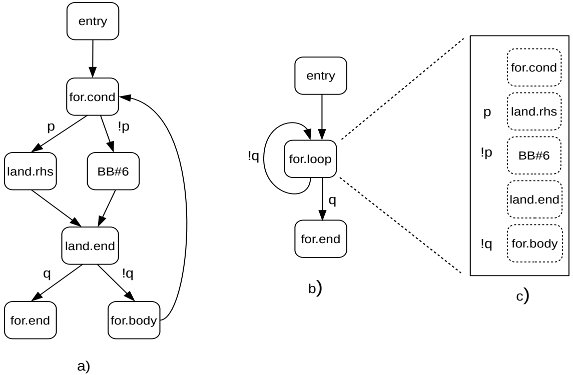


b)

Data transfer

- a) explicit, via inserted COPY instructions
- b) implicit, via intercluster crosspaths, 1 cycle delay (*crosspath stall*) for uses placed directly after definitions

If-Conversion



Basics

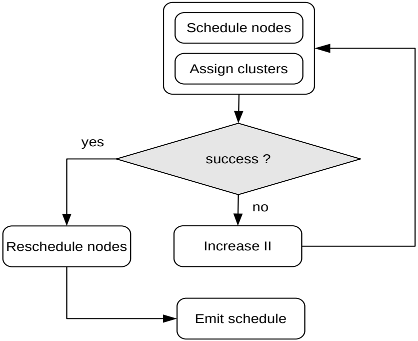
- As preprocessing to modulo scheduling
- Requires hardware support, removes conditional branches
- Reduces basic block count, increases ILP

Modulo scheduling (1)

General

- Iterative *II* scheme, swing scheduling adaptation
- Extended to address target specific factors such as functional unit support and crosspath stalls
- Employs modulo variable expansion based on lifetime analysis
- Utilizes modulo resource table, captures crosspath occupation

Modulo scheduling (2)



Specific

- Two-pass setup:
  - Iteratively generate a preliminary schedule in combination with provided clustering heuristics
  - Distribute intercluster copies, avoid crosspath stalls

Cluster assignment

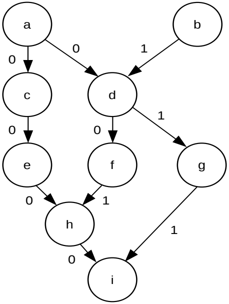
Simple naive heuristic

- Non-integrated, losely coupled with scheduling routine
- DG depth ordering, uniform handling of all dependences
- Processes the DG at once in a top-down manner
- Decides upon already assigned predecessor nodes only

Extended variant

- Runs inline with the modulo scheduler
- Operates on a DG with edges annotated prior to scheduling
- Uses a simple *copy cost* scheme for DG edge annotation
- Additionally incorporates cluster utilization counters

Copy-cost annotation



Details

- Qualifies adjacent nodes in terms of register copies
- Annotation only, no cluster information generated
- Takes crosspath access possibilities into account

Performance factors

Optimization objectives

- Fast schedule generation
- Minimal *initiation interval* through iterative scheme
- Reduction of *crosspath stalls* through explicit rescheduling
- Minimization of *intercluster copies* through DG labeling
- Even *cluster balance* through utilization counters

Performance evaluation: UAS as baseline

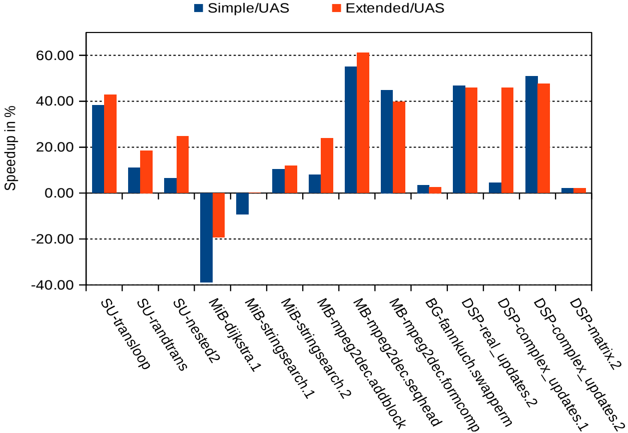


Figure: Cycle speedup (%) comparison to UAS

Performance evaluation: optimal ILP as baseline

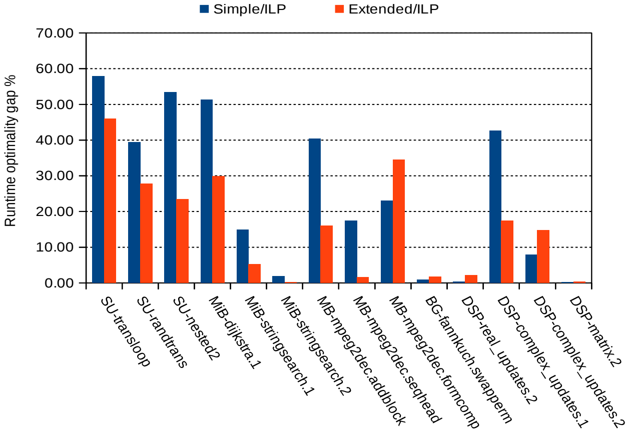


Figure: Optimality gap (%) to ILP

Performance evaluation: initiation intervals

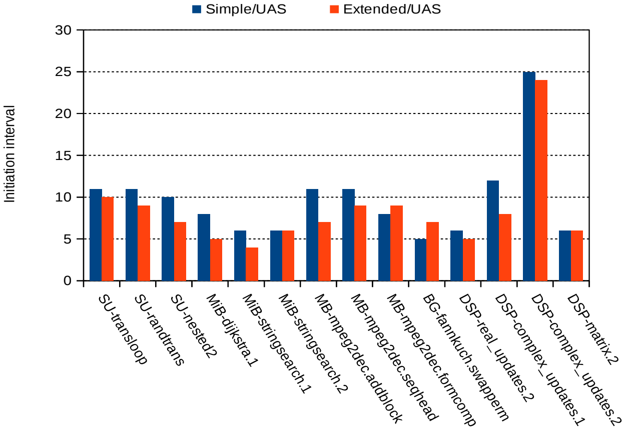


Figure: Absolute initiation interval values

Summary

Conclusions

- Extended clustering heuristic generally more potent
- Significant speedup compared to UAS (avg. 24.8%)
- Partially significant gap to ILP (avg. 15.8%)
- Nearly even cluster load distribution

Shortcomings, current research

- Backend modulo scheduling support currently very basic
- Rudimentary loop analysis, restricted applicability
- Clustering still suboptimal in terms of register copies
- More sophisticated clustering algorithms in development
- Fair, undistorted comparison to alternative implementations

Thank You

Thank you for being my audience!