Application Analysis with Integrated Identification of Complex Instructions for Configurable Processors

Nikolaos Kavvadias and Spiridon Nikolaidis

Section of Electronics and Computers,

Electronics Lab, Department of Physics,

Aristotle University of Thessaloniki,

54124 Thessaloniki, Greece

E-mail: **nkavv@skiathos.physics.auth.gr**

Introduction

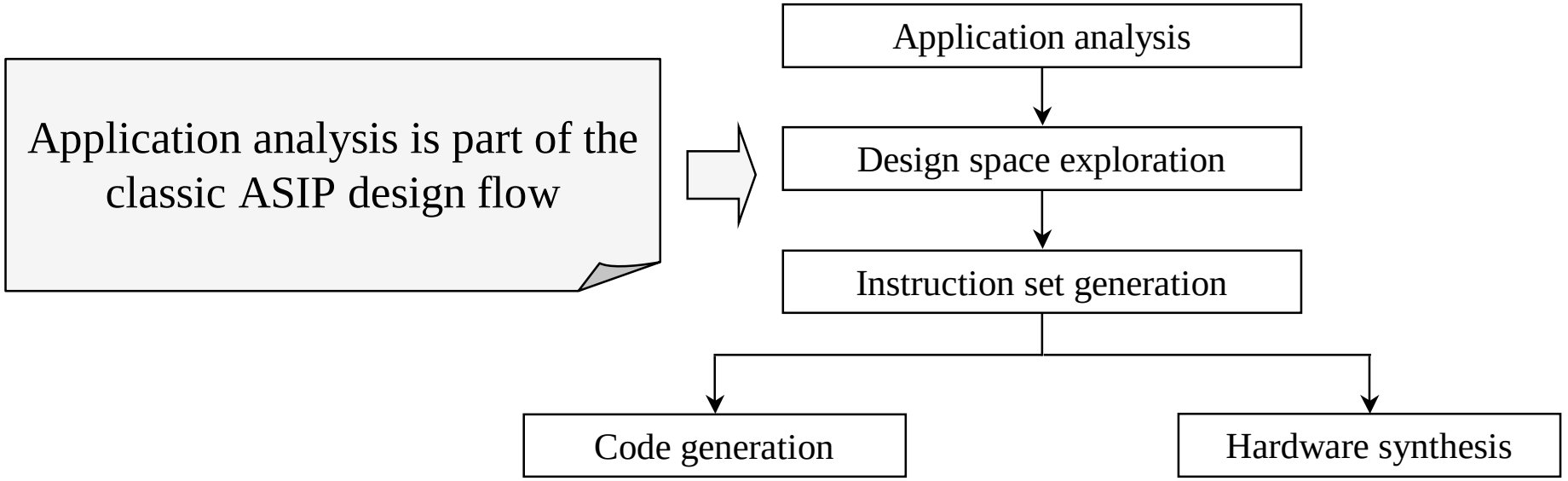
- Embedded processors present interesting architectural refinements in order to achieve performance/cost effective support of speed-critical applications e.g. video compression/decompression

- Customizable processors (for the embedded market) are a subset of application-specific instruction set processors (ASIPs) with reduced degrees of freedom in regard to application tuning
  - specific processor template is assumed, more stable retargetable software tools

- Recent customizable processors fall within this category, often marketed as ***configurable*** and ***extensible*** processors
  - **Configurability**: setting core parameters (e.g. cache size, multiplier topology etc), modifications within the architecture template
  - **Extensibility**: instruction-set enhancements by adding single-, multi-cycle and/or pipelined versions of complex instructions

The task of application analysis

- ***Application analysis***: obtain the profile of an application in respect to static and dynamic characteristics
  - **Static characteristics**: average basic block size, static instruction mix, static branch probabilities, register liveness, natural loop analysis
  - **Dynamic characteristics**: basic block execution frequencies, dynamic instruction mix, ratio of address to data computations



Key issues to proficient application analysis

- Application analysis guides the processor designer to decide on the appropriate configuration and instruction extensions for a given cycle performance/area constraint tradeoff and certain power budget

- ***Candidate instruction identification*** integrated to application analysis can assist in restricting (*pruning*) the exploration space for faster evaluation of possible solutions

- The required information is already available, early in the exploration flow, as directed-acyclic graphs (DAGs) of nodes in the application's CFG
  - An initial exploration pass on the performance-critical DAGs helps focusing on the performance improvement which usually has a positive side-effect to energy consumption

Requirements of a retargetable framework for design space exploration of processors

- Development of *tools* (assembler, linker, compiler backend, cycle-accurate simulator, debugger)
- The most critical is the optimizing compiler backend

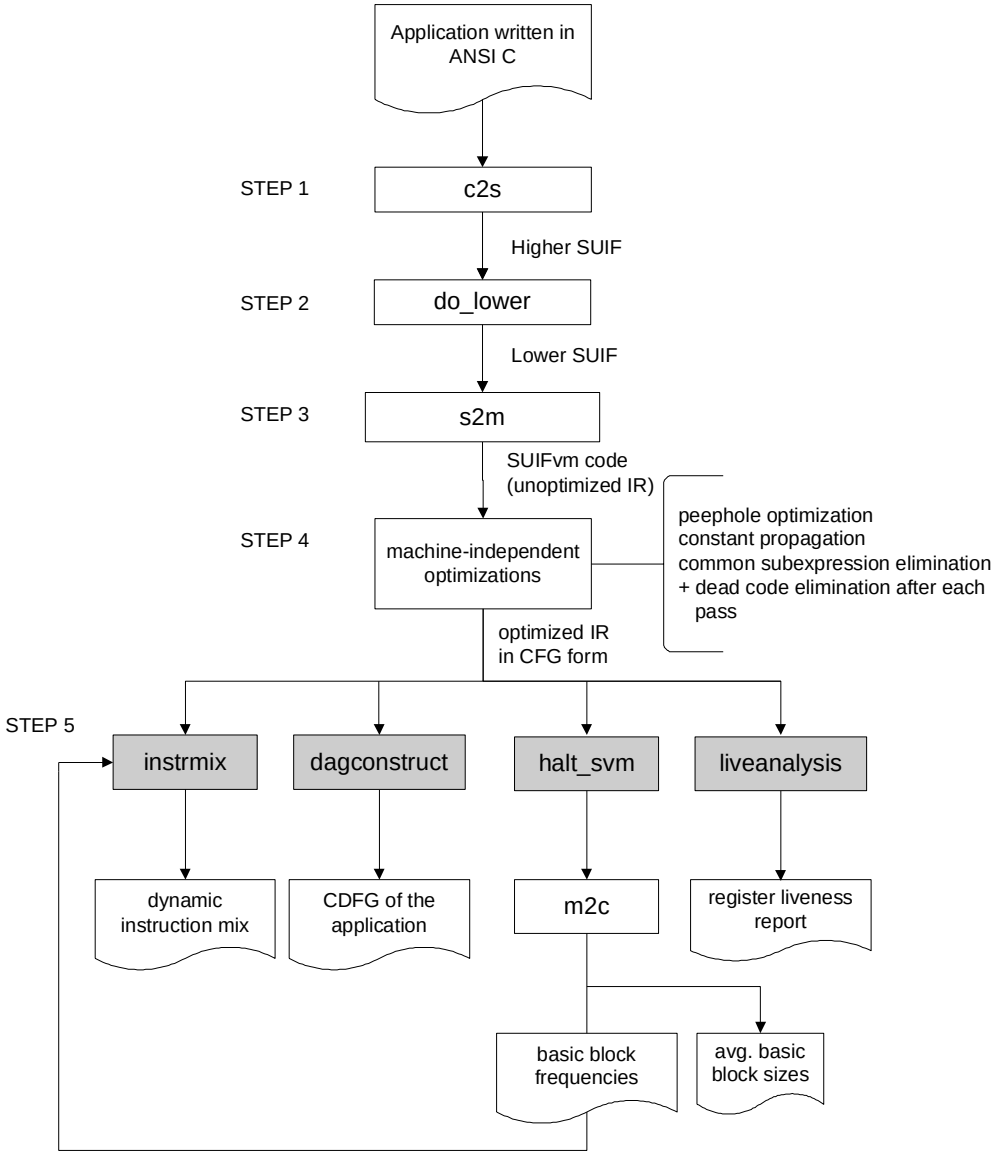
- A brief summary of (open licensing) retargetable compiler options
  - **GCC**: lots of host/target combinations, powerful optimizations, production-level code, non-modular approach thus very difficult to extent
  - **LCC**: lightweight, considerable quantity of backends, usable for simple RISCs, there is not enough infrastructure to extend!
  - **PROMIS**: parallelizing compiler for VLIW research, alpha state
  - **IMPACT**: stable retargetable VLIW compiler, moderate-to-significantly difficult to extend
  - **SUIF/MachSUIF**: a research frontend/backend infrastructure, small number of backends, difficult to add new backends, **but** very clean API for new analysis and optimization passes

MachSUIF "built-in" architecture/instruction set template for design space exploration

- Due to the difficulty of producing new robust backends in MachSUIF, most researchers are driven to one of the following two choices
  - Use either a x86-type (x86*, ia32, ia64) or most often the Alpha target which is a RISC target with similarities to both MIPS and SPARC v7/v8
  - Use the SUIFvm (SUIF virtual machine) instruction set as a "symbolic" target

- ***SUIFvm*** is the SUIF compiler IR (intermediate representation) with consistent 3-address code characteristics
- While missing certain features (e.g. procedure enrtry/exit instruction sequences) it incorporates the application's behavior
- **Reasonable** since at early stage of design space exploration, details of the hardware architecture are often not known (exploration/ optimization variants)

The proposed approach for application analysis and characterization



Base SUIF/MachSUIF passes:

1. C parsing
2. frontend analysis
3. generation of SUIFmv instructions

Analysis passes (non-existent in distributed MachSUIF):

-Static/dynamic instruction mix

-DAG construction (uses ***cfg***)

-Basic block frequencies (***halt***)

-Liveness analyzer (uses ***cfa***)

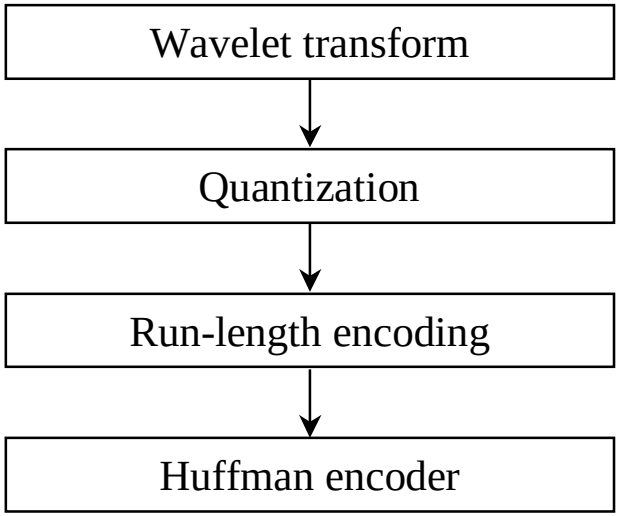
Potential traps and pitfalls in application characterization with MachSUIF

- Major drawback in the MachSUIF approach to application characterization was the difficulty in obtaining “quality code”
- Architecture-independent optimizations producing unexpectedly poor results
  - e.g. unnecessary type casting operations added after each optimization pass
- The problem of CSE (common-subexpression elimination)
  - Its usefulness in regard to candidate instruction matching depends on the algorithm applied to complex instruction generation
  - Case A: overlapped templates are permitted, then CSE will not prohibit the identification of beneficial candidates
  - Case B: orthogonal covers are used, and operator sharing possibilities will be missed due to eliminating a certain subexpression
- Note that register allocation has not been applied, but this is in favor of finding the authentic data flow dependencies implied by the algorithm [ClarkN03]

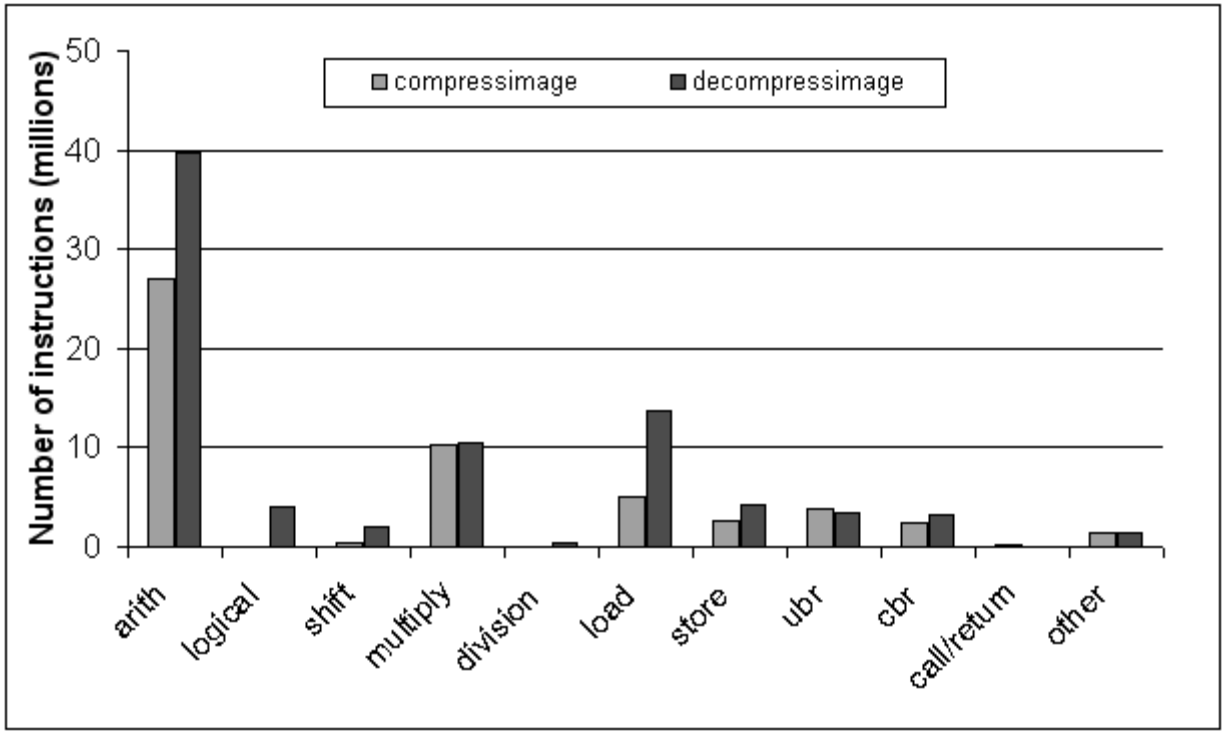
Case study: Wavelet/Scalar Quantization (WSQ) Image Compression algorithm

- Case study: The WSQ image compression algorithm, which is part of the Adaptive Computing Benchmarks [KumarS00]
- It is used for fingerprint image compression
- Both encoding and decoding algorithms have been evaluated with our MachSUIF-based framework

Processing flow of the encoder part for the WSQ image compression algorithm



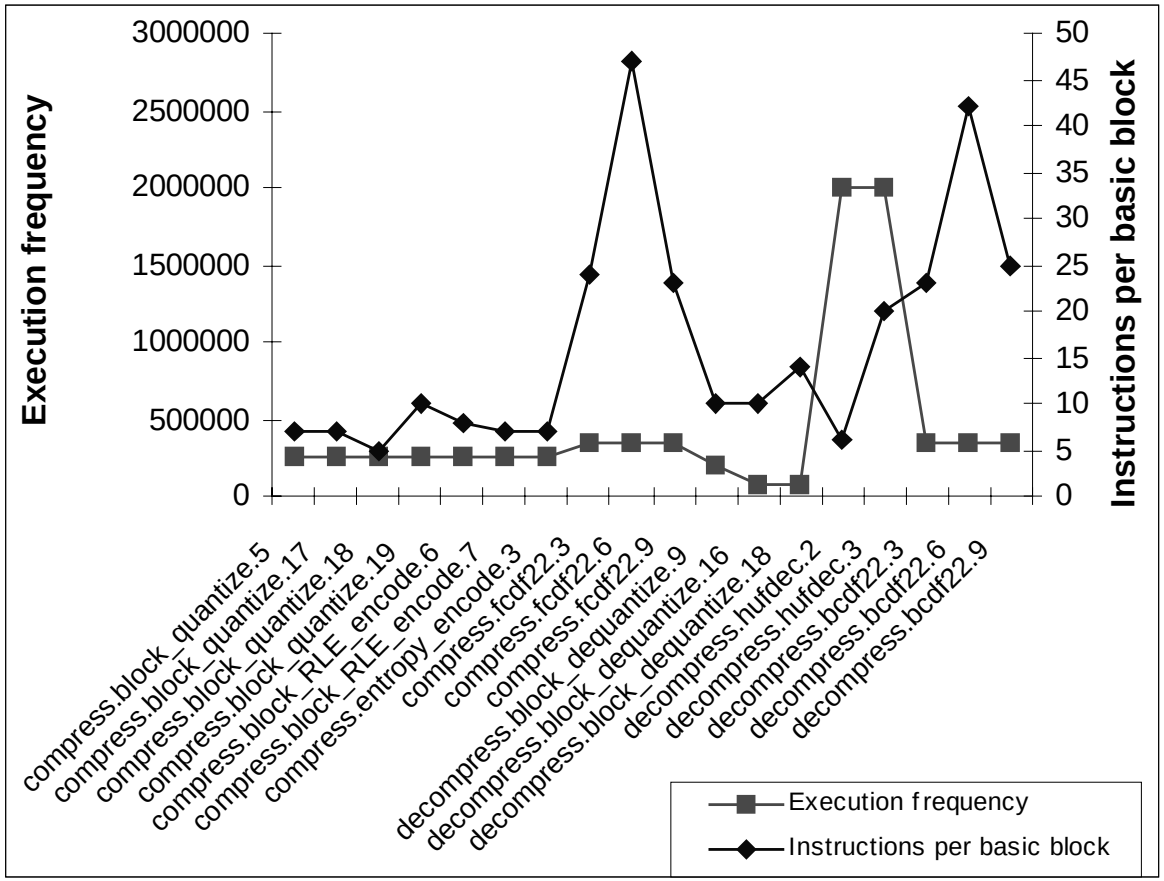
Instruction mix characteristics of WSQ encoding/decoding algorithms



WSQ is a pure integer application

- As expected for wavelet-based transforms, the computational complexity of the encoding and decoding algorithm are about the same
- Small ratio of branches to total instructions (9.8%). Even though the avg. basic block sizes are 3.98 and 4.48 respectively, some of the larger BBs have the most significant contribution

Basic block frequencies



- Each BB is assigned a unique name of the form:
  `<file_name>.<function_name>.<basic_block_number>`
- 10 BBs are responsible for 85.3% of dynamic instructions in ***compress*** and 8 BBs for the corresponding 97.3% in ***decompress***

Proposed set of characterization metrics for identifying candidate instruction extensions

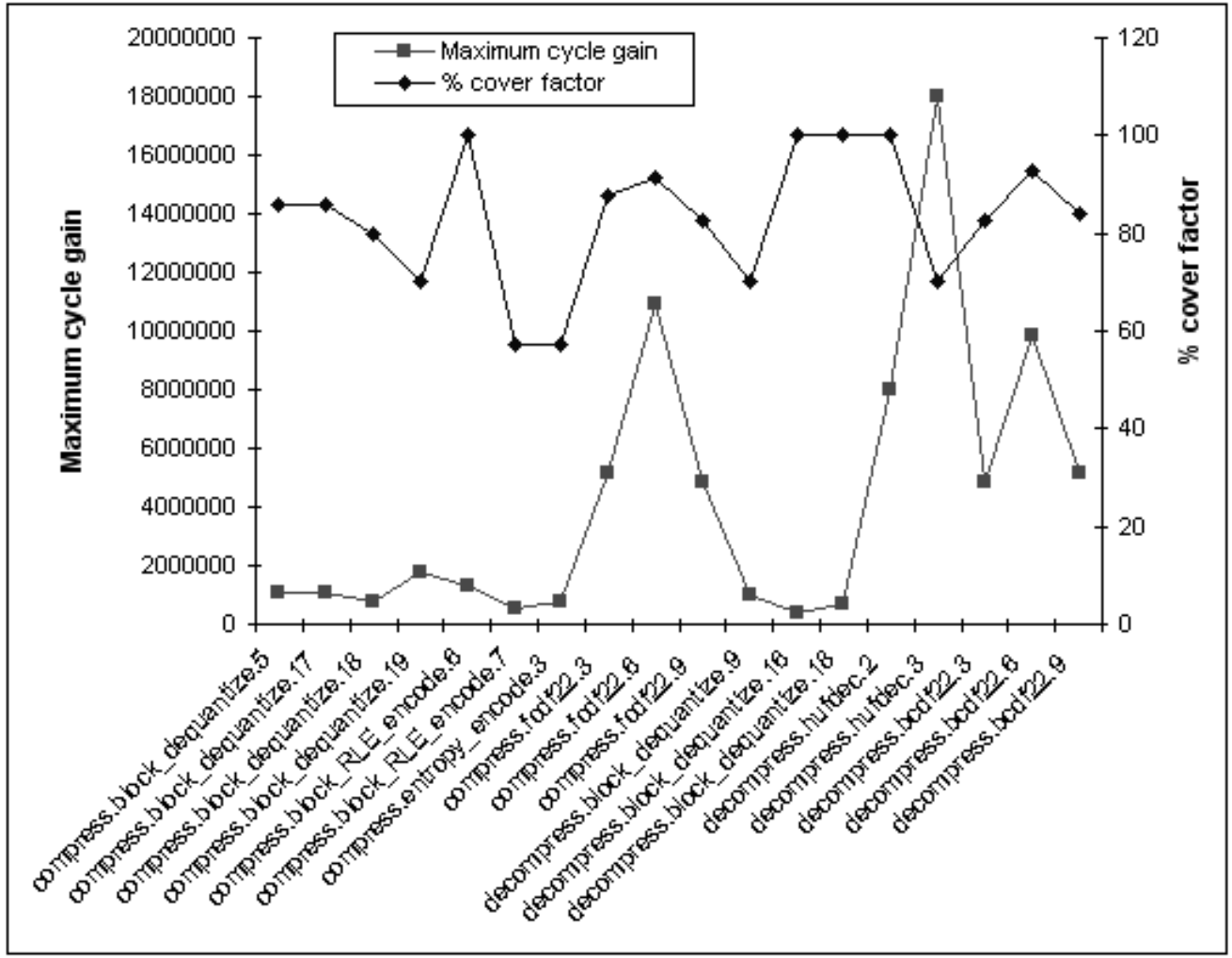
- Three new quantities have been proposed in context of integrating candidate instruction generation to conventional application analysis

- ***Percentage cover factor***
  - Determined by the proportion of the number of instructions after selection to the number of instructions prior selection (static metric)

- ***Maximum basic block cycle gain***
  - Calculated as the product of the maximum performance gain in cycles (no data hazards and spills to memory) with the execution frequency of the specific basic block (dynamic metric)

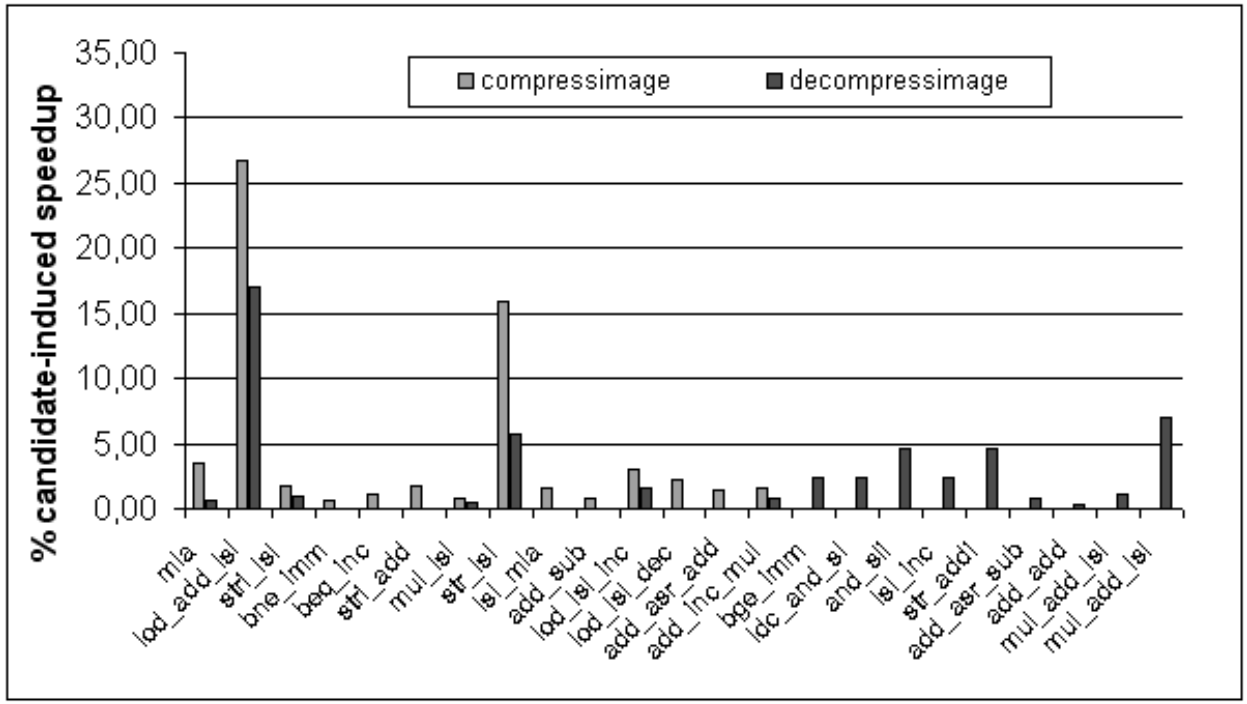
- ***Candidate-induced application speedup***
  - The application speedup due to selecting a complex instruction. Estimates the performance impact of selecting a set of isomorphic patterns.

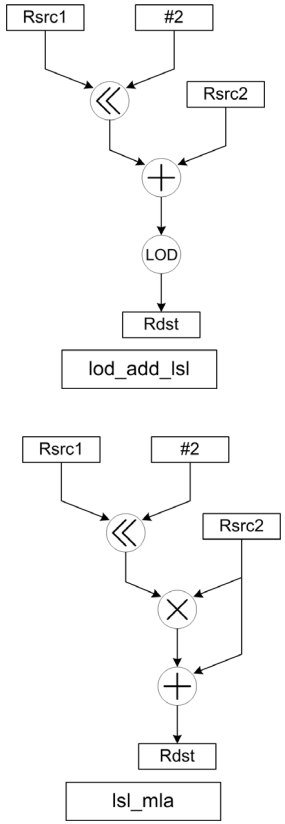
Template selection results for the performance-critical basic blocks



- Avg. percentage cover is 83.2%

Candidate-induced speedups for the *compress* and *decompress* applications





- 3/1 constraint on number of input/output operand was assumed for the complex instruction
- Complex load/store operations as *lod_add_lsl*, *str_lsl*, and *mul_add_lsl* implementing shifter-based addressing modes are good instruction candidates

Conclusions and future work

- An application analysis flow has been proposed for producing a set of complex instructions as a starting point to design space exploration iterations
- Novel application parameters have been introduced in the scope of application characterization:
  - ***Percentage basic block cover***
  - ***Maximum basic block cycle gain***
  - ***Candidate-induced application speedup***
- Estimated performance improvements in machine cycles: 52.9% for compress and 54% for decompress
- ***On-going work***: backend for a ***suifrm*** (real-machine) target, “textbook” as well as research instruction schedulers as optimization passes in MachSUIF, increasing the scope of candidate instruction for aggressive optimizations

References

[Gonzalez00] R. Gonzalez, “Xtensa: A configurable and extensible processor,” IEEE Micro, Vol. 20, No. 2, pp. 60-70, 2000.

[MachSUIF] M.D. Smith, G. Holloway, “An introduction to Machine SUIF and its portable libraries for analysis and optimization,” Tech. Rpt., Division of Eng. and Applied Sciences, Harvard University, 2.02.07.15 edition, 2002.

[GuptaTVK00] T.V.K. Gupta, P. Sharma, M. Balakrishnan, S. Malik, “Processor evaluation in an embedded systems design environment,” 13th Int. Conf. on VLSI Design, pp. 98-103, 2000.

[Leupers03] R. Leupers, O. Wahlen, M. Hohenauer, T. Kogel, P. Marwedel, “An executable intermediate representation for retargetable compilation and high-level code optimization,” Int. Wshp. on Systems, Architectures, Modeling and Simulation (SAMOS), Samos, Greece, 2003.

[ClarkN03] N. Clark, H. Zhong, W. Tang, S. Mahlke, “Automatic Design of Application Specific Instruction Set Extensions through Dataflow Graph Exploration,” Int. Journal of Parallel Programming, Vol. 31, No. 6, pp. 429-449, 2003.

[KumarS00] S. Kumar, L. Pires et al., “A benchmark suite for evaluating configurable computing systems – Status, reflections, and future directions,” ACM Int. Symp. on FPGAs, 2000.