Combining Source-Level Transformations and Instruction Set Extension

Alastair Murray[1], Richard Bennett[1],
Björn Franke[1], Nigel Topham[1]

* *Institute for Computing Systems Architecture (ICSA), School of Informatics, University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, EH9 3JZ, UK*

ABSTRACT

Industry's demand for flexible embedded solutions providing high performance and short time-to-market has led to the development of configurable and extensible processors. These pre-verified baseline cores allow for some degree of customisation through user-defined instruction set extensions (ISE). The traditional design flow for ISE is based on plain C sources of the target application and, after some ISE identification and synthesis stages, a modified source file is produced with explicit handles to the new machine instructions. We combined the exploration of source-level transformations and ISE identification to explore potential benefits. We show that the instruction extensions generated by automated tools are heavily influenced by source code structure. Our results demonstrate that a combination of source-level transformations and instruction set extensions can yield average performance improvements of 47%. This outperforms instruction set extensions when applied in isolation, and in extreme cases yields a speedup of 2.85. This work is also described in more detail in [1].

KEYWORDS: Customisable Processors, ASIPs, Source-Level Transformations, Compilers, Instruction Set Extension, Design Space Exploration

1 Introduction

High performance and short time-to-market are two of the major factors in embedded systems design. In recent years, processor IP vendors have addressed these by developing configurable and extensible processors such as the ARC 600 and 700, Tensilica Xtensa, ARM OptimoDE, and the MIPS Pro. These cores provide system designers with pre-verified solutions, thus reducing the risks and costs of a new processor design. Large degrees of flexibility are offered through instruction set extensions (ISE), which may help improve performance of compute-intensive kernels.

In order to explore different ISEs during the design stage and to trade off various, partially contradictory, design goals (e.g. performance, power, chip area) tool support is indispensable. Existing commercial (e.g. [2]) and academic (e.g. [3]) tools analyse an application written in C, identify can-

[1]E-mail: {r.v.bennett,a.c.murray}@sms.ed.ac.uk, {bfranke,npt}@inf.ed.ac.uk

didate instruction templates, modify the application's source code and insert handles to the newly created instructions. In general, the overall effectiveness of this approach depends on the designer's ability to generate complex instruction templates that (a) can replace a sufficiently large number of simple machine instructions, (b) are frequently executed and (c) can be efficiently implemented. We address problems (a) and (b), and show that the selection of "good" instruction templates is strongly dependent on the shape of the C code presented to ISE generation tool.

2 Methodology

Our methodology (illustrated in Figure 1) uses a probabilistic search algorithm and a source-level transformation tool to generate many different – but semantically equivalent – versions of the input program. We present each generated version to our ISE tool, which creates a set of new instructions and profile-based estimates of the speed-ups gained by using these instructions.

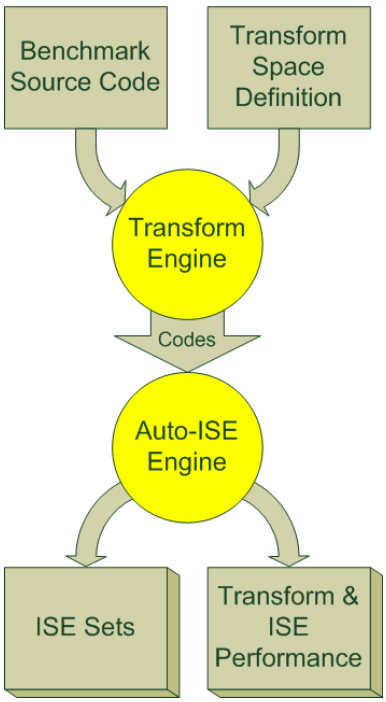

Figure 1: The combined but phased searching of transform and ISE design spaces.

To represent the transformation design space in this experiment, we use a source to source transformation tool built upon the SUIF1 [7] compiler framework. Samples are taken with uniform probability, at random point across the entire space of potential transformation. A sample in this sense is an ordered set of transformations and 10000 samples are taken per benchmark.

The set of transformed source codes forms a representative sampling of the entire search space for that benchmark. Each sample is then processed by an automated profiling ISE tool based on the Atasu et al. Integer Linear Programming method of derivation [4] built within the CoSy [5] compiler system. The tool operates in three phases: instrumentation, execution of instrumented binary and extension. During the extension phase the top four most beneficial instructions are taken and estimated speed-ups are calculated based on the profile using baseline processor instruction latencies from an Intel XScale PXA270 processor.

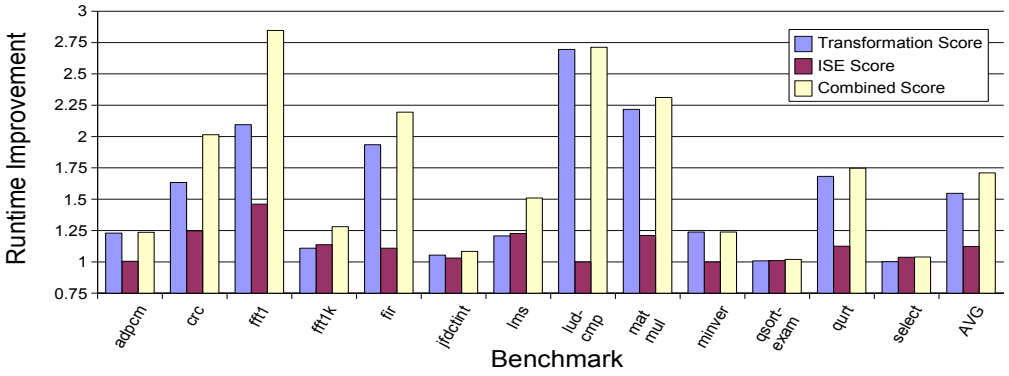

Figure 2: Runtime Improvements achieved on the SNU-RT benchmarks.

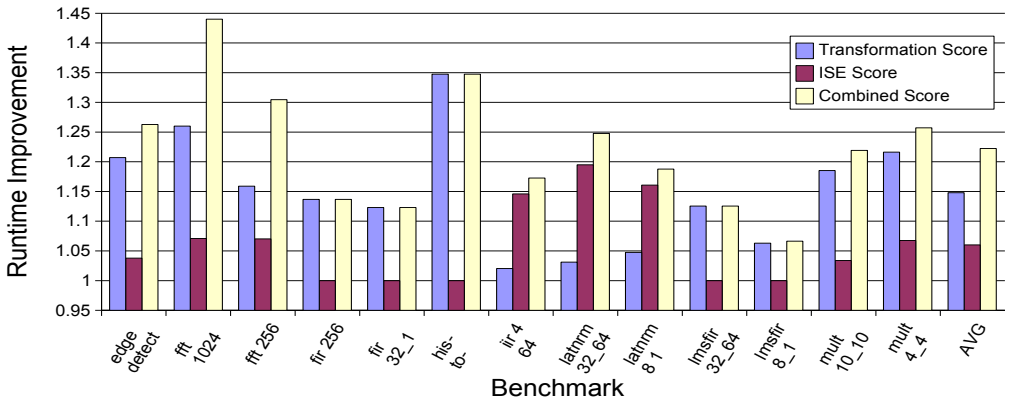

Figure 3: Runtime Improvements achieved on the UTDSP benchmarks.

The benchmarks used were taken from the SNU-RT [8] and UTDSP [9] suites:

- **SNU-RT**; adpcm, crc, fft1, fft1k, fir, jfdctint, lms, ludcmp, matmul, minver, qsort-exam, qurt, select.
- **UTDSP**; edge_detect, fft_1024, fft_256, fir_256, fir_32_1, histogram, iir_4_64, latnrm_32_64, latnrm_8_1, lmsfir_32_64, lmsfir_8_1, mult_10_10, mult_4_4.

3 Results

Figures 2 and 3 show the runtime improvements achieved on a selection of benchmarks from the SNU-RT and UTDSP suites. For each benchmark the three bars represent the best improvement seen in the search space for each technique. Peak runtime improvements of 2.70x (SNU-RT ludcmp), 1.46x and 2.85x (both SNU-RT FFT) are seen, for transformations alone, ISE alone and the combination of the two, respectively. Average runtime improvements across both benchmark suites are 1.35x, 1.09x and 1.47x respectively. It can also be seen that of the 26 benchmarks considered, 5 see a combined transformation and ISE runtime performance improvement of over 2.0x and only 6 see an improvement of less than 1.15x.

Examples where combined performance is significantly better than either transformations or ISE alone are SNU-RT CRC and the FFTs in both suites.

4 Conclusions

We have described a methodology for improved ISE generation that combines the exploration of high-level source transformations and low-level ISE identification. We have demonstrated that source-to-source transformations are not only very effective on their own, but provide much larger scope for performance improvement through ISE generation than any other isolated low-level technique. We have integrated both source-level transformations and ISE generation in a unified framework that can efficiently optimise both hardware and software design spaces for extensible processors.

We have experimentally demonstrated that our approach gives an average speedup of 1.47x. Compared to previous work [6], we have covered a much broader array of existing transformations to get a more global picture of the potential for transformation in improving instruction set extension.

Future work will investigate the integration of machine learning techniques based on program features into our design space exploration algorithm and target a commercial extensible processor platform.

References

[1] R. Bennett, A. Murray, B. Franke, and N. Topham. Combining source-to-source transformations and processor instruction set extension for the automated design-space exploration of embedded systems In *Proceedings of the ACM SIGPLAN/SIGBED 2007 Conference on Languages Compilers, and Tools for Embedded Systems (LCTES)*, 2007.

[2] ARC International. ARChitect product brief, 2007.

[3] R. Leupers, K. Karuri, S. Kraemer, and M. Pandey. A design flow for configurable embedded processors based on optimized instruction set extension synthesis. In *Proceedings of Design Automation & Test in Europe (DATE)*, Munich, Germany, 2006.

[4] K. Atasu, G. Dundar, and C. Ozturan. An integer linear programming approach for identifying instruction-set extensions. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '05)*, 2005.

[5] *ACE CoSy Website - http://www.ace.nl/compiler/cosy.html*.

[6] P. Bonzini, and L. Pozzi. Code transformation strategies for extensible embedded processors. In *CASES '06: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 242–252, New York, NY, USA, 2006. ACM Press.

[7] R. Wilson, R. French, C. Wilson, S. Amarasinghe, J. Anderson, S. Tjiang, S. Liao, C-W. Tseng, M. Hall, M. Lam, and J. Hennessy. SUIF: An infrastructure for research on parallelizing and optimizing compilers. *SIGPLAN Notices*, 29(12), 1994.

[8] *SNU-RT Real-Time Benchmarks - http://archi.snu.ac.kr/realtime/benchmark/*.

[9] C. Lee. *UTDSP Benchmarks - http://www.eecg.toronto.edu/ corinna/DSP/infrastructure/UTDSP.html*, 1998.