mal model of cache conflict misses based on a conflict graph which is enriched by data stemming from static WCET analysis. Since it is likely that this conflict graph initially contains many superfluous edges that do not model actual cache conflicts, this paper also proposes techniques to refine the conflict graph. These refinement approaches rely on control flow analyses, cache may analyses and on the inspection of a program's memory layout.

On top of this conflict graph model, a greedy heuristic for code positioning is presented. In contrast to previously published positioning approaches, our technique is inherently able to apply code positioning both to basic blocks and to entire functions. Furthermore, we are able to optimize for broad classes of cache architectures with varying associativities. The effectiveness of our approach is shown by average reductions of accumulated cache conflict misses by 15.5% and by average reductions of WCET estimates by 6.1% for an industrially relevant processor with a 2-way set-associative cache. Considering a direct-mapped cache even yields significantly larger savings in terms of both accumulated cache misses and WCET estimates.

Despite the fact that the average runtimes of 240 CPU seconds required by our optimization are still moderate, we are aware of the inherent scalability issues of an optimization heuristic where a full static WCET analysis is performed during each individual iteration of the heuristic's optimization loop. For this reason, our future work will concentrate on approximations of a WCET timing model which are faster than a fully-featured WCET analysis using *aiT*, but which still are accurate enough to guide our code positioning heuristics towards a systematic WCET reduction. Furthermore, we will apply our code positioning techniques to large industrial benchmarks in the future.

Acknowledgments

The authors would like to thank AbsInt Angewandte Informatik GmbH for their support concerning WCET analysis using *aiT*. The authors would also like to thank Synopsys for the provision of the instruction set simulator *CoMET* enabling the determination of ACETs.

7. REFERENCES

[1] AbsInt Angewandte Informatik GmbH. aiT: Worst-Case Execution Time Analyzers. `http://www.absint.com/ait`, July 2011.

[2] H. Falk and J. C. Kleinsorge. Optimal Static WCET-aware Scratchpad Allocation of Program Code. In *Proceedings of DAC*, July 2009.

[3] H. Falk and P. Lokuciejewski. A compiler framework for the reduction of worst-case execution times. *Real-Time Systems*, 46(2), Oct. 2010.

[4] H. Falk, S. Plazar, and H. Theiling. Compile-Time Decided Instruction Cache Locking Using Worst-Case Execution Paths. In *Proceedings of CODES+ISSS*, Oct. 2007.

[5] C. Ferdinand, F. Martin, and R. Wilhelm. Applying Compiler Techniques to Cache Behavior Prediction. In *Proceedings of LCT-RTS*, June 1997.

[6] C. Ferdinand and R. Wilhelm. Efficient and Precise Cache Behavior Prediction for Real-Time Systems. *Real-Time Systems*, 17(2), Nov. 1999.

[7] G. Gebhard and S. Altmeyer. Optimal Task Placement to Improve Cache Performance. In *Proceedings of EMSOFT*, Sept. 2007.

[8] C. Guillon, F. Rastello, T. Bidault, et al. Procedure Placement using Temporal-Ordering Information: dealing with Code Size Expansion. In *Proceedings of CASES*, Sept. 2004.

[9] Informatik Centrum Dortmund e. V. ICD-C Compiler framework. `http://www.icd.de/es/icd-c`, July 2011.

[10] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of MICRO*, Washington DC, Dec. 1997.

[11] E. A. Lee. Absolutely Positively On Time: What Would It Take? *Embedded Systems Column, IEEE Computer*, July 2005.

[12] Y. Liang and T. Mitra. Improved Procedure Placement for Set Associative Caches. In *Proceedings of CASES*, Oct. 2010.

[13] T. Liu, M. Li, and C. J. Xue. Minimizing WCET for Real-Time Embedded Systems via Static Instruction Cache Locking. In *Proceedings of RTAS*, Apr. 2009.

[14] P. Lokuciejewski, H. Falk, and P. Marwedel. WCET-driven Cache-based Procedure Positioning Optimizations. In *Proceedings of ECRTS*, July 2008.

[15] Mälardalen WCET Research Group. WCET Benchmarks. `http://www.mrtc.mdh.se/projects/wcet/benchmarks.html`, July 2011.

[16] S. Plazar, P. Lokuciejewski, and P. Marwedel. WCET-aware Software Based Cache Partitioning for Multi-Task Real-Time Systems. In *Proceedings of WCET*, June 2009.

[17] S. Plazar, P. Lokuciejewski, and P. Marwedel. WCET-driven Cache-aware Memory Content Selection. In *Proceedings of ISORC*, May 2010.

[18] I. Puaut and C. Pais. Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. In *Proceedings of DATE*, Apr. 2007.

[19] J. Reineke, D. Grund, C. Berg, et al. Timing Predictability of Cache Replacement Policies. *Real-Time Systems*, 37(2), Nov. 2007.

[20] S. Steinke, L. Wehmeyer, B.-S. Lee, et al. Assigning Program and Data Objects to Scratchpad for Energy Reduction. In *Proceedings of DATE*, Mar. 2002.

[21] V. Suhendra, T. Mitra, A. Roychoudhury, et al. WCET Centric Data Allocation to Scratchpad Memory. In *Proceedings of RTSS*, Dec. 2005.

[22] Synopsys, Inc. `http://www.synopsys.com`, July 2011.

[23] UTDSP Benchmark Suite. `http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html`, July 2011.

[24] X. Vera, B. Lisper, and J. Xue. Data Cache Locking for Higher Program Predictability. In *Proceedings of SIGMETRICS*, June 2003.

[25] M. Verma, L. Wehmeyer, and P. Marwedel. Cache-Aware Scratchpad Allocation Algorithm. In *Proceedings of DATE*, Feb. 2004.

[26] V. Živojnović, J. M. Velarde, C. Schläger, et al. DSPstone: A DSP-Oriented Benchmarking Methodology. In *Proceedings of ICSPAT*, Dallas, USA, Oct. 1994.

[27] WCET-aware Compilation. `http://ls12-www.cs.tu-dortmund.de/research/activities/wcc`, July 2011.

[28] W. Zhao, D. Whalley, C. Healy, et al. Improving WCET by Applying a WC Code-Positioning Optimization. *ACM Transactions on Architecture and Code Optimization*, 2(4), Dec. 2005.