Influence of Onchip Scratchpad Memories on WCET prediction[*]

Lars Wehmeyer, Peter Marwedel

Embedded Systems Group, CS Dept., University of Dortmund, Germany

{Lars.Wehmeyer, Peter.Marwedel}@udo.edu

May 27, 2004

Abstract

In contrast to standard PCs and many high-performance computer systems, systems that have to meet real-time requirements usually do not feature caches, since caches primarily improve the average case performance, whereas their impact on WCET is generally hard to predict. Especially in embedded systems, scratchpad memories have become popular. Since these small, fast memories can be controlled by the programmer or the compiler, their behavior is perfectly predictable. In this paper, we study for the first time the impact of scratchpad memories on worst case execution time (WCET) prediction. Our results indicate that scratchpads can significantly improve WCET at no extra analysis cost.

1 Introduction

For the currently available technologies, there is an increasing speed gap between processor speeds and memory speeds. Caches are being used in order to bridge that gap, especially in PC-like systems. However, the approach used in such systems has some disadvantages for embedded systems:

1. Caches are known to be one of the main contributors to the total energy consumption of systems [1], and

2. Caches are typically designed to improve the average case access time.

Analysis techniques to determine their contribution to the worst case execution time are complicated and, for some replacement policies, just missing. Scratch pad memories (sometimes also known as tightly coupled memories) are small memories mapped into the address space of a system. They are used whenever an address is within the address range assigned to that memory. Scratch pad memories are more energy efficient than main memories (since they are smaller) but also more efficient than caches (since only the required information is read from or written into the scratchpad memory). Scratchpads are currently being used by designers in a very ad-hoc fashion, and a comprehensive methodology of how to use them is, surprisingly, still missing.

Earlier work proposed compile-time algorithms for mapping hot spots of applications to scratchpad memories. This work was mainly motivated by the resulting energy savings, much of which result from a reduction of the average access time. However, the algorithms also have an extremely beneficial impact on worst case execution time estimation: it is fully predictable which memory will be used for a certain memory access. Hence, scratchpad memories provide 100% predictability concerning the timing of memory references. This predictability is explored in the current paper. In this work, we combine views from three different perspectives: an architectural view on scratchpad-based memory structures, a compiler view on how to map hot spots to these memories and a real-time system view on the resulting WCET. To the best of our knowledge, it is the first paper that provides a detailed analysis of the impact on the WCET of optimized mappings of applications to scratchpad memories.

2 Related Work

Many architectural features are included in modern microprocessors in order to meet the customers' demand for high average case performance. Especially in embedded systems having to meet real-time constraints, this is in general not very helpful, since the inclusion of pipelines, caches and branch prediction units makes it more difficult to predict a guaranteed upper bound for worst case exeution time [2]. Complicated analysis tools have been developed and are in use to shed light on the effect of these architectural features on WCET (see [3] for an overview). The difficulty lies in the fact that e.g. for caches, the hardware detects at runtime whether a memory access results in a cache hit or miss. In order to predict this behavior during WCET analysis, the worst case behavior of the cache has to be determined for the considered application. Several analysis methods have been proposed for instruction caches [4, 5] as well as for data caches [6]. The aforementioned publications solely deal with inclusion of caches in WCET estimation, which shows the considerable analysis effort required to predict cache behavior.

---

aiT [7] is a software tool that can help developers of safety-critical applications to verify that their programs will always meet the specified deadlines. This is done by determining an upper bound for the worst case execution time of the application. aiT guarantees the generated WCET results to be safe, which is generally infeasible using simulation techniques alone. Also, aiT abolishes the need to perform time consuming simulation runs in order to determine typical performance values. The aiT WCET analyzer has been designed according to the requirements of Airbus France for validating the timing behaviour of critical avionics software.

Scratchpad memories are being used as an alternative to caches due to their performance and their reduced energy consumption [8]. Scratchpad memories do not have a hardware to control their contents at runtime. Therefore, the assignment of memory objects to the different memories has to be handled either by the programmer or, in an automated process, by the compiler, who can analyze memory access patterns and distribute objects accordingly. The scratchpad can either retain the assigned memory objects throughout the running time of the application (static case), or the contents of the scratchpad may change at runtime (dynamic case). Allocation techniques to statically allocate data to the scratchpad were introduced e.g. in [9], whereas [10, 11] presented a dynamic approach for data and instructions, respectively. Further work concerning the utilization of scratchpad memories was conducted by [12, 13]. Both static and dynamic scratchpad usage are under full control of the compiler or the programmer, making the methods inherently predictable at compile time. In this paper, we will concentrate on the static allocation technique.

For the work presented in [14], the goal of the static allocation of both instructions and data to the scratchpad memory is energy saving. Therefore, an instruction level energy model for the used processor, an ARM7TDMI [15], was developed [16] and used in the *encc* compiler. The compiler determines the execution counts of functions and basic blocks and the number of accesses to variables in order to compute the most promising objects to be assigned to the limited scratchpad space. The actual optimization problem, which is similar to the well-known knapsack problem, is solved using an ILP solver. Then, the chosen memory objects are placed on the scratchpad, making control flow and address corrections where necessary.

3 Workflow

To determine a scratchpad memory's impact on WCET, we used the workflow shown in figure 1: The *encc* compiler generates an executable program which makes use of the available scratchpad. The memory objects allocated to the scratchpad memory are chosen according to the following algorithm (for details, please refer to [14]) which selects those elements with the highest benefit with respect to energy. In order to do this, all memory objects are weighted according to their execution or access frequency (for functions or data elements, respectively). The size of the objects is also considered, allowing the opti-
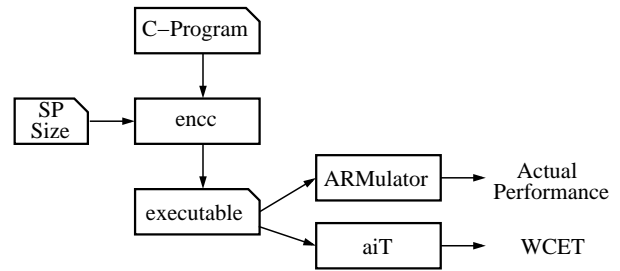

Figure 1: Workflow

mzation problem to be formulated as an integer programming problem as follows:

$$Maximize \sum_i m(i) * E(F_i) + \sum_j m(j) * E(Data_j)$$

subject to

$$\sum_i m(i) * S(F_i) + \sum_j m(j) * S(Data_j) <= SPsize$$

where $m(x)$ is a binary decision variable having the value $'1'$ if the corresponding object is allocated to the scratchpad and $E(x)$ is the benefit in energy consumption if object x is stored on the scratchpad instead of main memory. $S(x)$, the size of object x, is used in the constraints to ensure that the scratchpad capacity is not exceeded. In this form, the optimization problem can be solved using a commercial ILP solver [17]. The *encc* compiler then uses the solver's results to allocate the chosen objects to the scratchpad memory. The scratchpad size is varied in powers of two from 0 to a total of 4096 bytes in our experiments.

The generated executable with the optimal set of objects allocated to the scratchpad is then fed through ARM's instruction set simulator *ARMulator* to obtain the number of actually executed cycles for the given input data set. Apart from this, the executable is analyzed using aiT [7] to determine an upper bound for the WCET (commonly called WCET) of the executable.

aiT supports the specification of memory regions with different attributes. The only relevant attribute for this work is the number of wait states that occur during memory accesses. According to the values measured for our ARM7 evaluation board, we assumed three waitstates for all main memory accesses and one wait state for scratchpad accesses.

To enable aiT to analyze the executable with memory objects allocated to the scratchpad, some annotations concerning instructions that use PC-relative addressing are required. These annotations ensure that the correct addresses will always be assumed during aiT's analysis. In order to keep the manual annotation overhead low, we decided to allow only the allocation of complete functions (i.e. not basic blocks and multi basic blocks as described in [14]) and data elements onto the scratchpad. This restriction can be easily overcome with a slightly increased annotation effort. The used toolchain supports this annotation process.

4 Results

The benchmarks used to explore the impact of a scratchpad on WCET are given in table 1. They comprise two speech encoding and decoding algorithms from the mediabench benchmark suite [18]. The programs were compiled with varying scratchpad sizes, as described in the previous section. The execution time is expected to decrease (along with the energy consumption) when the scratchpad capacity is increased. The effect of larger scratchpad size on average case performance and on WCET can be seen in figures 2 to 4.

Name	Description
adpcm	Speech encoding and decoding using Adaptive Diff. Pulse Code Modulation
G.721	Speech encoding and decoding, reference implementation of the CCITT
Multi_Sort	Combination of sorting algorithms

Table 1: Benchmarks

Execution Time vs. WCET (G.721)

Figure 2: Results for G.721 benchmark

The G.721 benchmark takes a little more than 2 million cycles to complete with our used input data on a system with only one main memory, whereas aiT estimates the WCET for the worst case input to be about 4 million cycles. Since it is in general not possible to determine the worst case input data set for an arbitrary application, using a simulation approach is not feasible to determine a guaranteed upper bound for WCET. As can be seen in figure 2, increasing the scratchpad capacity not only improves the average execution time, but also has a strong positive effect on the WCET estimate. Where average case execution time is reduced to about 1,250,000 cycles for a 4k scratchpad, corresponding to a reduction of 43%, WCET reduces down to 1,650,000 cycles, which means a reduction of 58% compared to the inital case with no scratchpad. Thus, the effect on WCET is even greater than the effect on average case execution time.

For the Multi_Sort benchmark, similar results can be observed. By only changing the scratchpad capacity and using our compile-time algorithm to solve the problem of allocating an optimal set of memory objects to the scratchpad memory,
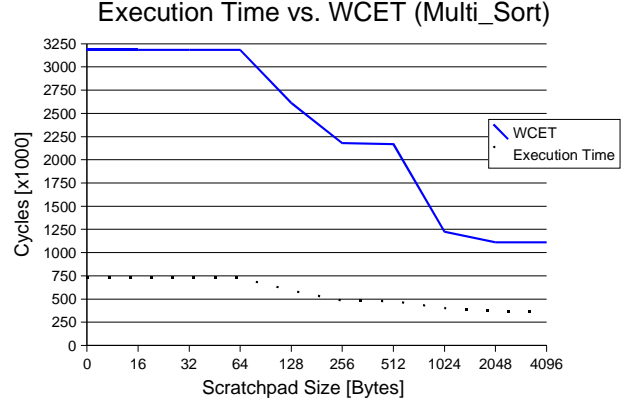
Execution Time vs. WCET (Multi_Sort)

Figure 3: Results for Multi_Sort benchmark

we find that the WCET decreases by about 65%, whereas the actual execution time for our used average input data only decreases by about 50%. Without further requirements concerning WCET analysis (as e.g. required if a cache was used in the system), the use of a scratchpad memory thus shows a positive impact on WCET.

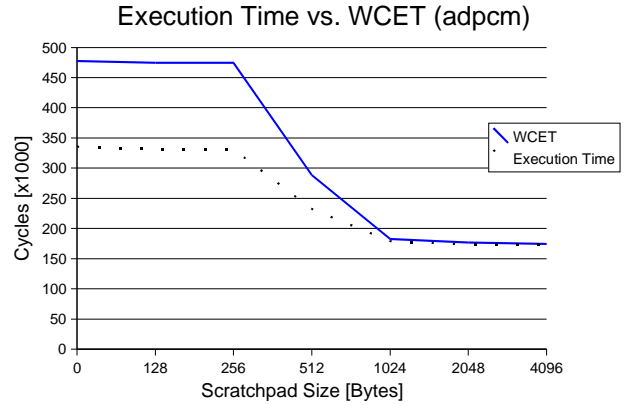Execution Time vs. WCET (adpcm)

Figure 4: Results for adpcm benchmark

For the adpcm benchmark, even the initial WCET values are very close to the actual execution times. This seems to be due to the fact that all execution paths within this benchmark are very similar to the critical path. Despite this good initial WCET estimate, using a scratchpad can still improve the WCET prediction: If an onchip memory of more than 512 bytes is used, the difference between actual performance and WCET becomes negligible. The reductions in average case execution and WCET estimate reduce by 49% and 63%, respectively, highlighting the fact that WCET benefits strongly from use of a scratchpad memory.

One reason for the positive effect of scratchpad memories is possibly due to worst case assumptions concerning pipeline stalls. In the case of a three stage pipeline (as in the ARM7TDMI used in our experiments), a pipeline stall will require three instructions to be fetched from memory before the next result is generated by the CPU. If the latency for a single memory access is three cycles, then nine additional memory

cycles will be required to completely re-fill the pipeline (assuming, as on our evaluation board, memory chips that do not support accelerated burst transfers). If, on the other hand, the used memory has a latency of only one cycle (as is the case for a scratchpad memory), then the pipeline can be filled with only three additional memory cycles. aiT has to assume all possible pipeline stalls to be able to guarantee that the predicted WCET result is always safe. The fact that the overhead for these pipeline stalls can be reduced by using a scratchpad memory explains the good results concerning WCET.

5 Summary and Future Work

In this work we show for the first time that using scratchpad memories in real-time systems is beneficial for WCET estimation. Using a known algorithm to allocate memory objects (both instructions and data) to the scratchpad memory, and a commercially available WCET analysis tool, we have shown that the decrease of the WCET caused by scratchpad memories is even larger than the decrease of the average case execution time. This is possible without any modification in the used timing analysis tool. Many performance enhancing architectural modifications (e.g. caches) make WCET estimation a difficult task. If scratchpads are being used, the user only needs to know the latency cycles of the used memories.

This work shows an additional advantage of scratchpad based architectures beyond previously published results (which investigated average case execution time and energy consumption). All this is feasible with a decreased complexity of WCET tools.

In the future, we will consider how scratchpad memories compare to cache models that are being supported in some WCET analyzers today. This comparison is not really fair, since caches require extensive support and careful analysis in WCET analysis, whereas scratchpad memories can be integrated at no extra analysis costs. However, caches are being used in many systems today to improve the average performance and therefore have a practical significance.

Apart from using the energy-aware allocation algorithm from [14], we will also consider employing a similar technique which primarily takes into account the memory objects that lie on a program's critical path. By reinforcing the selection of these memory objects instead of those memory objects that consume most energy, the positive effect on WCET should become even more obvious.

6 Acknowledgement

References

[1] Milind B. Kamble and Kanad Ghose. Analytical Energy Dissipation Models for Low-Power Caches. In *Proc. International Symposium on Low Power Electronics and Des ign*, pages 143–148. ACM/IEEE, August 1997.

[2] Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The Influence of Processor Architecture on the Design and the Results of WCET Tools. *Proceedings of the IEEE*, 91(7), July 2003.

[3] Peter Puschner and Alan Burns. A Review of Worst-Case Execution-Time Analysis. *Journal of Real-Time Systems*, 18(2/3):115–128, May 2000.

[4] Yau-Tsun Steven Li, Sharad Malik, and Andrew Wolfe. Performance Estimation of Embedded Software with Instruction Cache Modeling. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 380–387, November 1995.

[5] Yau-Tsun Steven Li, Sharad Malik, and Andrew Wolfe. Cache Modeling for Real-Time Software: Beyond Direct Mapped Instruction Caches. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1996.

[6] Thomas Lundqvist. A WCET Analysis Method for Pipelined Microprocessors with Cache Memories. Technical report, Dept. of Computer Engineering, Chalmers University of Technology, June 2002.

[7] AbsInt Angewandte Informatik GmbH. aiT: Worst Case Execution Time Analyzers. http://www.absint.com/ait, 2004.

[8] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad Memory: A Design Alternative for Cache On-chip Memory in Embedded Systems. In *10th Int. Symp. on Hardware/Software Codesign (CODES)*, May 2002.

[9] P. R. Panda, N. D. Dutt, and A. Nicolau. *Memory Issues in Embedded Systems-On-Chip*. Kluwer Academic Publishers, 1999.

[10] M.Kandemir, J.Ramanujam, M.J.Irwin, N.Vijaykrishnanand I.Kadayif, and A.Parikh. Dynamic Management of Scratch-Pad Memory Space. In *Proceedings of the 2001 ACM Design Automation Conference*. DAC, June 2001.

[11] S. Steinke, N. Grunwald, L. Wehmeyer, R. Banakar, M. Balakrishnan, and P. Marwedel. Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory. *Int. Symp. on System Synthesis (ISSS)*, pages 213–218, 2002.

[12] J.Ph. Diguet, S. Wuytack, F. Catthoor, and H. De Man. Formalized Methodology for Data Reuse Exploration in Hierarchical Memory Mappings. In *ISLPED 1997 Monterey CA*. ACM, August 1997.

[13] P.R.Panda, F.Catthoor, N.D.Dutt, K.Danckaert, E.Brockmeyer, C.Kulkarni, A.Vandercapelle, and P.G.Kjeldsberg. Data and memory optimization techniques for embedded systems. pages 149–206, April 2001.

[14] S. Steinke, L.Wehmeyer, B.-S. Lee, and P. Marwedel. Assigning Program and Data Objects to Scratchpad for Energy Reduction. *Design, Automation and Test in Europe (DATE)*, pages 409–417, 2002.

[15] ARM Ltd. ARM7TDMI Technical Reference Manual. http://www.arm.com/pdfs/DDI0210B_7TDMI_R4.pdf, 2004.

[16] S. Steinke, M. Knauer, L. Wehmeyer, , and P. Marwedel. An Accurate and Fine Grain Instruction-Level Energy Model Supporting Optimizations. In *Proceedings of the International Workshop - Power and Timing Modeling, Optimization and Simulation, Yverdon-les-bains, Switzerland*, September 2001.

[17] ILOG. CPLEX. http://www.ilog.com/products/cplex.

[18] Stephen Brown. MediaBench Home. http://cares.icsl.ucla.edu/MediaBench, 2004.