IS CHIP-MULTIPROCESSING THE END OF REAL-TIME SCHEDULING?

Martin Schoeberl and Peter Puschner[1]

Abstract

Chip-multiprocessing is considered the future path for performance enhancements in computer architecture. Eight processor cores on a single chip are state-of-the art and several hundreds of cores on a single die are expected in the near future. General purpose computing is facing the challenge how to use the many cores. However, in embedded real-time systems thread-level parallelism is naturally used. In this paper we assume a system where we can dedicate a single core for each thread. In that case classic real-time scheduling disappears. However, the threads, running on their dedicated core, still compete for a shared resource, the main memory. A time-sliced memory arbiter is used to avoid timing influences between threads. The schedule of the arbiter is integrated into the worst-case execution time (WCET) analysis. The WCET results are used as a feedback to regenerate the arbiter schedule. Therefore, we schedule memory access instead of CPU time.

1. Introduction

Chip-multiprocessing (CMP) is considered non-trivial for real-time systems. Scheduling of multiprocessor systems is NP-hard. Standard CMP architectures communicate via cache coherence protocols for the level 1 caches. The resolution of conflicts due to the cache coherence protocol and memory access for the cache load and write back between unrelated tasks is practically intractable for the worst-case execution (WCET) analysis of single tasks. Therefore, we consider a time-predictable CMP system with a statically scheduled access to the main memory for real-time systems [12].

In this paper, we assume following simplification: we consider a CMP system where the number of processors is equal (or higher) than the number of tasks. In such a system classic CPU scheduling becomes a non-problem as each task runs on its own CPU. What we have to consider for the feasibility analysis is the access conflict to the shared resource main memory. Access to main memory has to be scheduled between the conflicting tasks. However, scheduling the memory access is different from scheduling of tasks in a uniprocessor system: The overhead of a *context switch* in a memory arbiter is almost zero. Therefore, fine-grained scheduling algorithms can be applied at this level.

We propose a fine-grained static schedule for the memory access. We include the knowledge of the possible memory access patterns that are determined by the memory schedule into the WCET analysis. Results from the WCET analysis are fed back into the assignment of the memory schedule. Therefore, in this system scheduling and WCET analysis play an interacting role for the real-time system design and the feasibility analysis.

[1]Institute of Computer Engineering, Vienna University of Technology, Austria;
email: mschoebe@mail.tuwien.ac.at, peter@vmars.tuwien.ac.at

ECRTS 2009
9th International Workshop on Worst-Case Execution Time (WCET) Analysis
http://drops.dagstuhl.de/opus/volltexte/2009/2288

1.1. Background

The evaluation of the proposed system is based on following technologies: the time-predictable Java processor JOP [16], a JOP CMP system with a time-sliced memory arbiter [12], and a WCET analysis tool for embedded Java processors [20] with an extension for CMP systems.

JOP is an implementation of the Java virtual machine (JVM) in hardware. JOP was designed as a real-time processor with time-predictable execution of Java bytecodes. One feature of JOP is the so called method cache [15]. The method cache simplifies the cache analysis by caching whole methods. The accurate cycle timing of JOP enabled the development of several WCET analysis tools for embedded Java [20, 4, 3, 5]. The timing model of JOP, which is used by the WCET analysis tools, simplifies our evaluation of the memory access scheduling for a time-predictable CMP. JOP is implemented in a field-programmable gate array (FPGA). Therefore, configurations for different applications and the task sets are a valuable option.

An extension of JOP to a time-predictable chip-multiprocessor system is presented in [12]. A time-sliced memory arbitration module allows execution of several threads on different cores without influencing each other's timing. The WCET analysis tool [20] has been adapted by Pitter [13] to include the instruction timings of memory accessing bytecodes for the CMP system. The version of the WCET analysis tool we use in our evaluation is an enhanced version of [20] that includes a tighter analysis of the method cache [5].

To the best of our knowledge, the JOP CMP system is the first time-predictable CMP that includes a WCET analysis tool. Although the presented CMP architecture is based on a Java processor, the main idea of a single task per core and scheduling of memory accesses instead of threads is independent of the application language.

1.2. Related Work

Chip-multiprocessor systems are state-of-the-art in desktop and server systems and are considered also for future embedded systems. Current processors contain up to eight cores [8, 6]. The Niagara T1 from Sun [8] is a design with 8 simple six-stage, single-issue pipelines similar to the original five-stage RISC pipeline. The additional pipeline stage adds fine-grained multithreading. Using single-issue in-order pipelines is good news for WCET analysis. However, the fine-grained multithreading and communication via a shared L2 cache make WCET analysis infeasible. In contrast to the T1 our CMP system has single threaded pipelines and avoids shared caches. All communication between the cores is performed via non-cached, shared memory.

The Cell multiprocessor [6] combines, besides a standard PowerPC microprocessor, 8 synergistic processors (SP) with 256 KB local memory. The cores are connected via an on-chip communication ring [7]. The SPs are in-order pipelines and can only access their local memory. Therefore, WCET analysis of SP programs should be straight forward. The main drawback of this architecture is that the communication between the cores (and main memory) has to be performed with explicit DMA transfers. Compared to that design we allow communication via shared memory without any restrictions.

Azul Systems provides an impressive multiprocessor system for transaction oriented server workloads [2]. A single Vega chip contains 54 64-bit RISC cores, optimized for the execution of Java programs.

Up to 16 Vega processors can be combined to a cache coherent multiprocessor system resulting in 864 processors cores and supporting up to 768 GB of shared memory.

The PRET project is devoted to the development of a time-predictable CMP system [11]. The processor cores are based on the SPARC V8 instruction set architecture with a six-stage pipeline and support chip level multithreading for six threads to eliminate data forwarding and branch prediction. An interesting feature of the PRET architecture is the *deadline* instruction that stalls the current thread until a deadline is reached. This instruction is used to perform time based, instead of lock based, synchronization for access to shared data. Besides using processor local scratchpad memories, the shared memory it accessed through a so called *memory wheel*. The memory wheel is a time division, multiple access (TDMA) based memory arbiter with equal slot slice. In contrast to the PRET project, we consider TDMA schedules that can be adapted to the application needs.

The approach, which is closest related to our work, is presented in [1, 14]. The CMP system is intended for tasks according to the simple task model [9]. In this model the task communication happens explicitly: input data is read at the begin of the task and output data is written at the end of a task. Furthermore, local cache loading for the cores is performed from a shared main memory. Similar to our approach, a TDMA based memory arbitration is used. The paper deals with optimization of the TDMA schedule to reduce the WCET of the tasks. The design also considers changes of the arbiter schedule during task execution. This implies that all tasks and the cores have to be synchronized with the memory arbiter, resulting in a cyclic executive model. All tasks, even on different cores, need to fit into a common major cycle.

In contrast to [1, 14], our approach to a TDMA based CMP system allows standard shared memory communication without the restrictions of a cyclic executive. Real-time tasks with arbitrary periods are supported. The approach in [1] uses explicit path enumeration to integrate the TDMA schedule into the WCET calculation, which is known to scale poorly. In our system the TDMA schedule is integrated into an implicit path enumeration based WCET tool [12]. Furthermore, a complete implementation of the proposed CMP system and the WCET analysis tool are available.

2. WCET Based Memory Access Scheduling

Embedded systems, also called cyber-physical systems, interact with the *real* world. And the real world, external to the computer system, is massively parallel. Therefore, embedded systems are good candidates for chip-multiprocessing. We argue that the transistors required to implement super-scalar architectures are better used on complete replication of simple cores for a CMP system [19].

CMP systems share the access bandwidth to the main memory. To build a time-predictable CMP system, we need to schedule the access to the main memory in a predictable way. A predictable scheduling can only be time based, where each core receives a fixed time slice. This scheduling scheme is called time division multiple access (TDMA). The execution time of loads, stores, and the cache miss penalty depend on this schedule. Therefore, the arbiter schedule has to be known to determine accurate WCET estimates.

Assuming that enough cores are available, we propose a CMP model with a single thread per processor. In that case thread switching and schedulability analysis for each individual core disappears. Since each processor executes only a single thread, the WCET of that thread can be as long as its deadline. When the period of a thread is equal to its deadline, 100% utilization of that core is feasible.

For threads that have enough slack time left, we can increase the WCET by decreasing their share of the bandwidth on the memory bus. Other threads with tighter deadlines can, in turn, use the freed bandwidth and run faster. The consumption of the access bandwidth to main memory is adjusted by the TDMA schedule. The TDMA schedule itself is the input for WCET analysis for all threads. Finding a TDMA schedule, where all tasks meet their deadlines, is thus an iterative optimization problem.

It can be argued that the assumption of having fewer tasks than available processing cores is too restrictive. However, we do not forbid task scheduling on a single core. Our approach is still valid when more demanding tasks *own* their core and other tasks share a core under preemptive scheduling.

Two benefits of the single task per core approach have to be emphasized. First, the CPU utilization on each core can reach 100%, even when the task periods are non-harmonic, and the system is still analyzable. Second, the overhead of task dispatching, and the hard to analyze cache thrashing, disappears. Both effects lead to tighter WCET bounds and can compensate for idle cycles on the cores when the tasks are waiting on their turn to access the memory.

2.1. Memory-Access Scheduling Schemes

Before we look for a particular memory-access schedule for an application running on a multiprocessor system, we have to decide about the general properties of the scheduling scheme.

- Granularity of the memory access scheduler: one has to decide on scheduling of single memory accesses versus scheduling of longer TDMA slots comprising a larger number of memory accesses. Shorter time slices provide the highest flexibility and the most fine-grained control in distributing memory bandwidth among the processors. Longer time slices may improve access times to shared memory if larger blocks from the shared memory have to be transferred at once.
- In case time slices of multiple memory accesses are used, one further has to decide whether time slices should be of uniform duration or different length. The use of equally sized time slices provides the simplest solution. Assigning different sizes to the tasks/CPUs makes a better adaption to application needs possible.
- Single scheduling pattern versus different scheduling patterns. For some applications, the memory access characteristics of the tasks do hardly ever change or the occurrence of particular access patterns is hard to predict. Such systems may use a single TDMA memory access pattern during their whole operational cycle. Other systems, in contrast, have a number of operational phases during which the need for memory bandwidth and memory access patterns of the different tasks/CPUs differ. Such systems benefit from a more flexible memory system that supports a number of schedules tailored to the needs of the tasks in the different phases of operation.
- Synchronous versus asynchronous operation of memory-access schedules and task execution. One of the most influential decisions on the design of the system is whether the TDMA memory accesses and the task execution cycles (i.e., activation times and periods of tasks) are synchronized or not. Synchronizing memory access schedules and task operation has the benefit that memory access schedules can be optimized to the needs of the tasks during their execution cycle, at the cost of a higher planning effort and a more rigid execution pattern at runtime. Unsynchronized operation is easier to plan, but allows for a less accurate adaption of task and memory-access behavior.

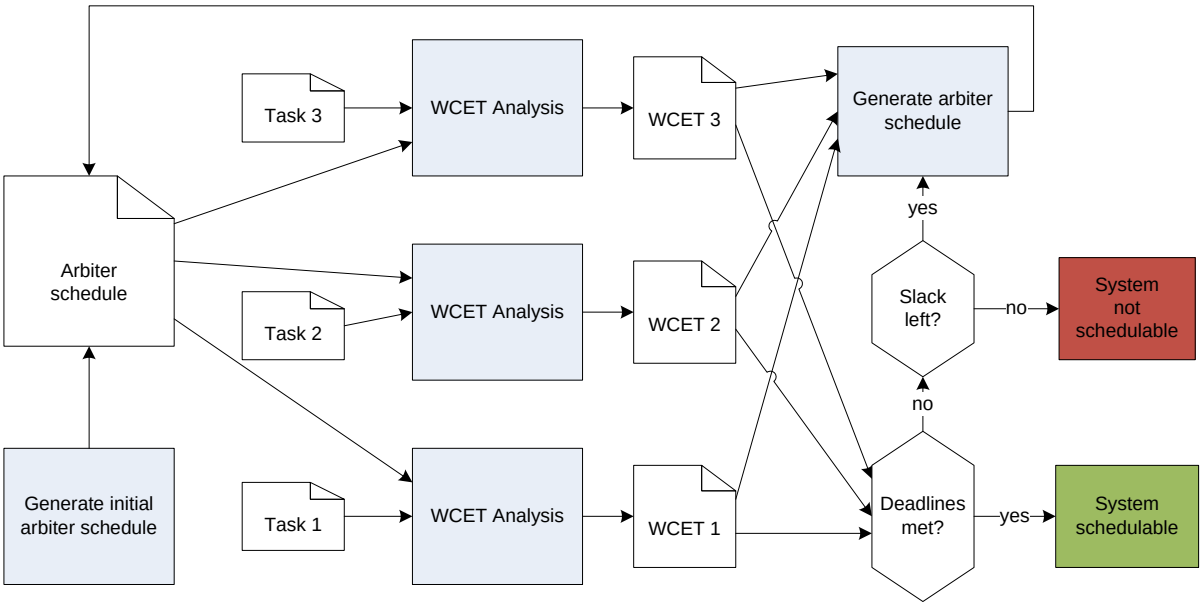

Figure 1. Tool flow for a CMP based real-time system with one task per core and a static arbiter schedule. If the deadlines are not met, the arbiter schedule is adapted according to the WCETs and deadlines of the tasks. After the update of the arbiter schedule the WCET of all tasks needs to be recalculated.

In the rest of the paper we assume that memory access schedules and task execution are not synchronized. Our goal is to find memory access schedules that split the memory bandwidth among the CPUs of the system in such a way that a maximum number of tasks of a given task set meet their timing requirements. We schedule single memory accesses and, at this time, assume that the same TDMA scheduling sequence is used throughout the whole time of system operation.

2.2. Tool Structure

Figure 1 shows the analysis tool flow for the proposed time-predictable CMP with three tasks. First, an initial arbiter schedule is generated, e.g., one with equal time slices. That schedule and the tasks are the input of WCET analysis performed for each task individually. If all tasks meet their deadlines with the resulting WCETs, the system is schedulable. If some tasks do not meet their deadline and other tasks have some slack time available, the arbiter scheduler is adapted accordingly. WCET analysis is repeated, with the new arbiter schedule, until all tasks meet their deadlines or no slack time for an adaption of the arbiter schedule is available. In the latter case no schedule for the system is found.

The TDMA schedule can be adapted along two dimensions. Firstly, the individual slot sizes can be varied. This variation can be very fine-grained down to single clock cycles. Longer slot sizes are mostly beneficial for burst transfers on a cache load. Secondly, the TDMA round can be adapted to include several (short) slots for a core within one TDMA round. This leads to lower WCET for individual memory loads and stores. Which adaption, or if a combination of the two approaches is beneficial depends on the application characteristic. In the following section both approaches are evaluated for a network application.

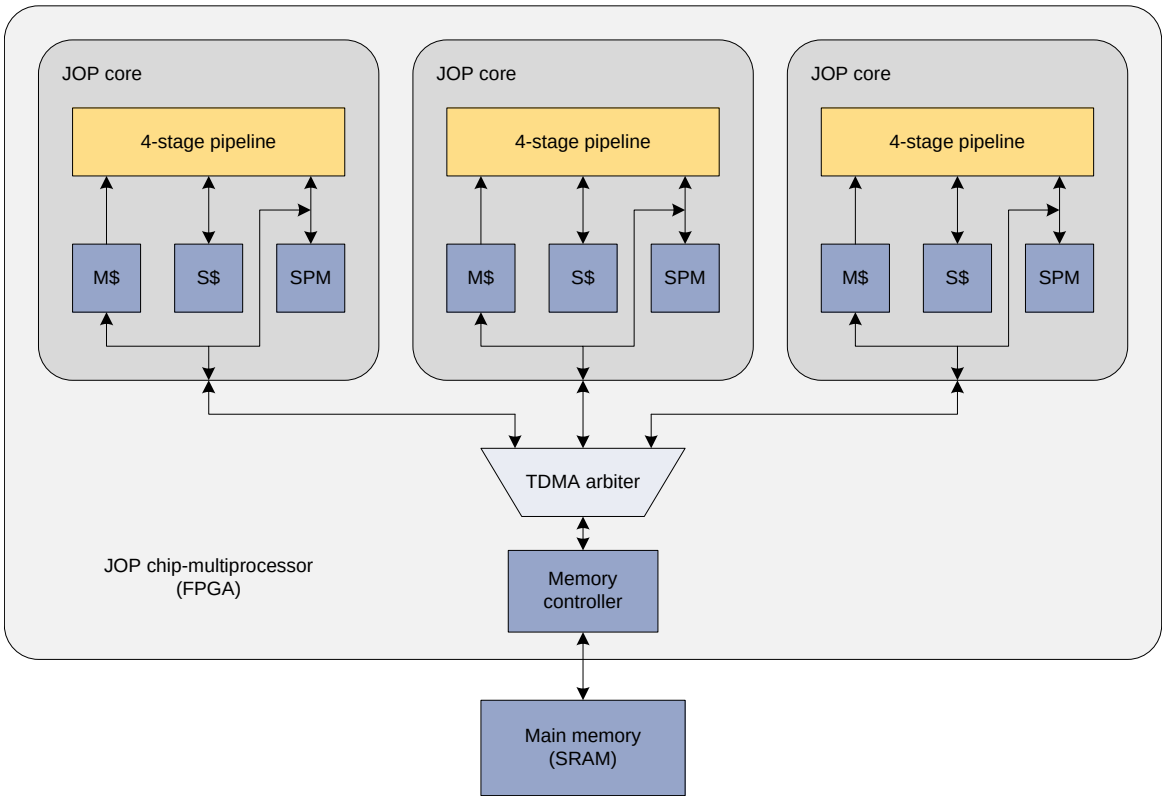

Figure 2. A JOP based CMP system with core local caches and scratchpad memories, a TDMA arbiter, the memory controller, and the shared main memory.

3. Evaluation

We evaluate our approach on the CMP version [13] of the Java processor JOP [16]. Figure 2 shows a configuration of a JOP CMP system with three cores. Each core contains a local method cache (M$), a local stack cache (S$), and a scratchpad memory (SPM). The local memories contain either thread local data (S$ and SPM) or read-only data (M$). Therefore, no cache coherency protocol is needed. The cores are connected to the TDMA based arbiter, which itself is connected to a memory controller. The memory controller interfaces to the shared, external memory.

For the WCET analysis and the schedule of the memory arbitration the memory timing is according to the implementation of the JOP CMP system on an Altera DE2 FPGA board [12]: 4 clock cycles for a 32-bit memory read and 6 clock cycles for a 32-bit memory write. The resulting minimum TDMA slot width is 6 clock cycles. The cache sizes for each core are 1 KB for the stack cache and 4 KB for the method cache.

3.1. The Baseline with the Example Application

As an example of an application with several communicating threads, we use an embedded TCP/IP stack for Java, called `ejip`. The benchmark explores the possibility of parallelization within the TCP/IP stack. The application that uses the TCP/IP stack is an artificial example of a client thread requesting a service (vector multiplication) from a server thread. That benchmark consists of 5 threads: 3 application threads (client, server, result), and 2 TCP/IP threads executing the link layer as well as the network layer protocol.

Function	Task	Single	5 cores	WCET increase
ipLink.run()	τ_1	1061	2525	2.4
resultServer()	τ_2	1526	3443	2.3
request()	τ_3	8825	23445	2.7
macServer()	τ_4	7004	17104	2.4
net.run()	τ_5	8188	18796	2.3
Sum		26604		

Table 1. WCET estimations in clock cycles for a single core and a 5 core system

One bottleneck in the original TCP/IP stack was a global buffer pool. All layers communicated via this single pool. The single pool is not an issue for a uniprocessor system, however real parallel threads compete more often for the access to the buffer pool. As a consequence, we have rewritten the TCP/IP stack to use dedicated, wait-free, single reader/writer queues [10] for the communication between the layers and the application tasks. The additional benefit of non-blocking communication is a simplification of the scheduling analysis, as we do not have to take blocking times into account.

For WCET analysis we use a new WCET tool for JOP [5] with an extension for TDMA memory arbitration [12]. The WCET tool performs the analysis at bytecode instruction level. The execution of bytecodes that access the main memory (e.g., field access, cache load on a method invoke or return) depends on the TDMA schedule. For those bytecodes, the worst-case phasing between the memory accesses and the TDMA schedule is considered for the low-level WCET timing.

Table 1 shows the WCET bounds of all 5 tasks obtained by our tool for a single core and a 5 core system connected to a TDMA arbiter with a slot size of 6 cycles for each CPU. Due to the memory access scheduling, the WCET increases for each individual task. Although 5 cores compete for the memory access the increase of the WCET is moderate: between a factor of 2.3 and 2.7. As 5 cores lead to an ideal speedup of a factor of 5 and the individual task are slower on that CMP (with respect to the analyzed WCET) by a factor of around 2.5, the net speedup of the 5 core CMP system is around a factor of $5/2.5 = 2$.

To simplify our example we assume that all tasks shall execute at the same period. As this example resembles a pipelined algorithm, this assumption is valid. With the default configuration τ_3 has the largest WCET and for a 100 MHz CMP system the minimum period is 235 µs. With this period the other tasks have a slack time between 47 µs and 210 µs. We use that slack time to adapt the TDMA schedule, as shown in Figure 1, to reduce the maximum period for all tasks. We have not yet implemented the automation of this process, but perform the arbiter schedule generation manually. The resulting WCETs for different arbiter schedules are shown in Table 2, with the equal schedule of 6 cycle slots for all cores in the second row for reference. A manual iteration of the arbiter schedule generation and WCET analysis loop took a few minutes. The WCET analysis, including data-flow analysis for loop bounds, takes around 27 s on a standard PC.

3.2. Variation of the Arbiter Schedule

For the first adaption of the arbiter schedule we assumed a quite naive approach: the slot size of the three longer tasks is doubled to 12 cycles. The result is shown in column *Schedule 1* of Table 2. The WCET for τ_4 and τ_5 has been reduced, but due to the longer TDMA round (48 cycles instead of 30

Task	Equal	Schedule 1	Schedule 2	Schedule 3	Schedule 4	Schedule 5	Schedule 6
τ_1	2525	3605	3605	5045	4325	5285	5525
τ_2	3443	4865	4865	6761	5813	7077	7393
τ_3	23445	25313	20211	20201	20473	17955	18681
τ_4	17104	15284	14860	19468	14276	16756	17376
τ_5	18796	17856	16432	14832	16204	18924	18364

Table 2. WCET estimations in clock cycles of the application tasks with different arbiter configurations

cycles) the WCET of task τ_3 actually increased – an effect into the wrong direction.

For *Schedule 2* and *Schedule 3* the basic slot size was kept to the minimum of 6 cycles, but different inter-leavings of the tasks are used: $< \tau_1, \tau_3, \tau_4, \tau_5, \tau_2, \tau_3, \tau_4, \tau_5 >$ and $< \tau_3, \tau_3, \tau_5, \tau_5, \tau_4, \tau_1, \tau_3, \tau_3, \tau_5, \tau_5, \tau_4, \tau_2 >$. For *Schedule 3* the repetition of two 6 cycle slots for the same task results effectively in a single 12 cycle slot. Both schedules decrease the execution time of task τ_3 to 202 µs at the expense of higher execution times of some other tasks.

The following experiments are based on *Schedule 2* with variations of the individual slot sizes. We use following notation for the schedule: τ_i/n is a slot of n cycles for task i. Schedule 4, $< \tau_1/6, \tau_3/8, \tau_4/8, \tau_5/8, \tau_2/6, \tau_3/8, \tau_4/8, \tau_5/8 >$ increases the slot size to 8 cycles for all, but tasks τ_1 and τ_2. The resulting minimum period, 205 µs, is similar to *Schedule 2*. To reduce the WCET of τ_3 we double the slot size for τ_3, resulting in Schedule 5, $< \tau_1/6, \tau_3/16, \tau_4/8, \tau_5/8, \tau_2/6, \tau_3/16, \tau_4/8, \tau_5/8 >$, with a minimum period, now determined by task τ_5, of 189 µs. After several additional iterations we ended up with *Schedule 6*, $< \tau_1/6, \tau_3/16, \tau_4/8, \tau_5/10, \tau_2/6, \tau_3/16, \tau_4/8, \tau_5/10 >$, providing the best balanced system for the three demanding tasks and a minimum period of 187 µs. Any further increase of individual slot sizes resulted in a higher execution time for τ_3, τ_4, and τ_5. Therefore, not the whole slack time of τ_1 and τ_2 could be distributed to the other three tasks.

3.3. Discussion

The decrease of the minimum period for all tasks from 235 µs to 187 µs due to the optimization of the memory arbiter schedule is probably not too exciting. However, we started our experiment with a task set that is already quite balanced: three tasks had a similar WCET, and only two tasks had a shorter WCET. The resulting slack time of two tasks can be distributed to the other three tasks.

An interesting question is how the WCET performance scales with a CMP system with a TDMA based memory arbitration? As we assumed the same period for all tasks we can perform a simple comparison: when executing the five tasks on a single core the cumulative WCET is the sum of the WCET values as given in Table 1: 26604, resulting in a minimum period of the system of 266 µs. The same task set on the 5 core CMP system with the optimized arbiter schedule results in a minimum period of 187 µs. Therefore, the speedup of the WCET due to a 5 core CMP is about 1.4. This speedup is moderate, but it ignores more complex task sets with different periods and the overhead of preemptive scheduling on a single core, which is not present in the single task per core setup.

This paper presents a first step towards scheduling of memory accesses instead of tasks. The WCET analysis considers the arbiter schedule only for memory accesses within a single bytecode. An improvement of the analysis that considers basic blocks for the interleaving of memory accesses with

the TDMA schedule can provide tighter WCET bounds.

With respect to the TDMA schedule we are currently restricted to quite regular schedules. An extension to irregular patterns is straight forward and will enable a finer tuning of the memory bandwidth requirements.

Furthermore, we have avoided caching of heap data to simplify the predictability of the system. As a result, each access to heap allocated data has a WCET of a complete TDMA round, which easily reaches cache miss latencies of current architectures. First ideas on time-predictable data caching emerge [18]. A small, fully associative data cache will enable analysis of some memory accesses to heap allocated data. Even a low analyzable hit rate of the data cache will improve the WCET of the tasks on the CMP system.

We are confident that the suggested improvements will reduce the WCET of tasks on a CMP system considerable and enable usage of time-predictable CMP systems in hard real-time systems.

4. Conclusion

In this paper, we presented the idea to build chip-multiprocessor real-time systems where each application task has its own CPU for execution. Therefore, task scheduling is completely avoided. The remaining shared resource, the main memory and the memory bandwidth, needs to be scheduled. A time-slicing arbitration of the memory access is the only time-predictable form for CMP systems. When the arbitration schedule is known in advance the worst-case memory access time is known and can be integrate into WCET analysis.

Furthermore, we have presented an iterative approach to build the schedule for the memory arbiter. Starting from equal sized time slices for each core/task a first WCET estimation for each task is calculated. Tasks that have a shorter WCET than their deadline can be punished by reducing their share of the memory bandwidth. That bandwidth can be used by tasks that do not (yet) meet their deadline. The updated arbitration schedule is again used as input for WCET analysis. We have demonstrated this iterative approach with an example application in the context of a Java processor based CMP system.

The processor design, the TDMA based CMP design, and the WCET analysis tool are open source under the GNU GPL. The sources are available via git[2] and the build instructions can be found in [17].

Acknowledgements

We thank Christof Pitter for the design and implementation of the TDMA arbiter and the integration into the WCET tool during his PhD thesis. Furthermore, we are thankful to Benedikt Huber who has redesigned the WCET analysis tool for JOP during his Master's thesis. The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement number 216682 (JEOPARD) and 214373 (Artist Design).

[2] `git clone git://www.soc.tuwien.ac.at/jop.git`

References

[1] ANDREI, A., ELES, P., PENG, Z., AND ROSEN, J. Predictable implementation of real-time applications on multiprocessor systems on chip. In *Proceedings of the 21st Intl. Conference on VLSI Design* (Jan. 2008), pp. 103–110.

[2] AZUL. Azul compute appliances. Whitepaper, 2009.

[3] BOGHOLM, T., KRAGH-HANSEN, H., OLSEN, P., THOMSEN, B., AND LARSEN, K. G. Model-based schedulability analysis of safety critical hard real-time java programs. In *Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems (JTRES 2008)* (New York, NY, USA, 2008), ACM, pp. 106–114.

[4] HARMON, T. *Interactive Worst-case Execution Time Analysis of Hard Real-time Systems*. PhD thesis, University of California, Irvine, 2009.

[5] HUBER, B. Worst-case execution time analysis for real-time Java. Master's thesis, Vienna University of Technology, Austria, 2009.

[6] KAHLE, J. A., DAY, M. N., HOFSTEE, H. P., JOHNS, C. R., MAEURER, T. R., AND SHIPPY, D. J. Introduction to the Cell multiprocessor. *j-IBM-JRD 49*, 4/5 (2005), 589–604.

[7] KISTLER, M., PERRONE, M., AND PETRINI, F. Cell multiprocessor communication network: Built for speed. *Micro, IEEE 26* (2006), 10–25.

[8] KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro 25*, 2 (2005), 21–29.

[9] KOPETZ, H. *Real-Time Systems*. Kluwer Academic, Boston, MA, USA, 1997.

[10] LAMPORT, L. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering 3*, 2 (1977), 125–143.

[11] LICKLY, B., LIU, I., KIM, S., PATEL, H. D., EDWARDS, S. A., AND LEE, E. A. Predictable programming on a precision timed architecture. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008)* (Atlanta, GA, USA, October 2008), E. R. Altman, Ed., ACM, pp. 137–146.

[12] PITTER, C. Time-predictable memory arbitration for a Java chip-multiprocessor. In *Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems (JTRES 2008)* (Santa Clara, USA, September 2008), ACM Press, pp. 115–122.

[13] PITTER, C. *Time-Predictable Java Chip-Multiprocessor*. PhD thesis, Vienna University of Technology, Austria, 2009.

[14] ROSEN, J., ANDREI, A., ELES, P., AND PENG, Z. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Proceedings of the Real-Time Systems Symposium (RTSS 2007)* (Dec. 2007), pp. 49–60.

[15] SCHOEBERL, M. A time predictable instruction cache for a Java processor. In *On the Move to Meaningful Internet Systems 2004: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2004)* (Agia Napa, Cyprus, October 2004), vol. 3292 of *LNCS*, Springer, pp. 371–382.

[16] SCHOEBERL, M. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture 54/1–2* (2008), 265–286.

[17] SCHOEBERL, M. *JOP Reference Handbook*. No. ISBN 978-1438239699. CreateSpace, 2009. Available at `http://www.jopdesign.com/doc/handbook.pdf`.

[18] SCHOEBERL, M. Time-predictable cache organization. In *Proceedings of the First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009)* (Tokyo, Japan, March 2009), IEEE Computer Society.

[19] SCHOEBERL, M. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems vol. 2009, Article ID 758480* (2009), 17 pages.

[20] SCHOEBERL, M., AND PEDERSEN, R. WCET analysis for a Java processor. In *Proceedings of the 4th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2006)* (New York, NY, USA, 2006), ACM Press, pp. 202–211.