Using State of the Art Multiprocessor Systems as Real–Time Systems —
The RECOMS Software Architecture *

Jürgen Stohr Alexander von Bülow Georg Färber

Institute for Real–Time Computer Systems
Prof. Dr.–Ing. Georg Färber
Technische Universität München, Germany

Abstract

In this paper a software architecture for using a general purpose multiprocessor system based on the Intel x86 architecture as a real–time system is presented. This approach allows the execution of standard and real–time tasks in parallel. It is possible to separate and to minimize the influences of the hardware on worst–case execution times of real–time software. They are mainly caused by the processor architecture, the chip set and the bus systems being used. Compared to single processor systems, computation times become more deterministic and response times are shorter.

1. Introduction

Modern general purpose computer systems are not designed to act as hard real–time systems. The focus of optimization of theses systems is to provide a good average computing power. The minimization of maximum execution times of software remains unconsidered, as in most cases the users will not notice them. Optimization techniques which improve average performance but may result in poor worst case behavior are for example the use of caches, pipelines and the use of fairness communication protocols. However, general purpose computer systems are very low priced compared to other architectures, combined with a high computing power which makes them interesting for real–time usage. In addition, a computer system based on the Intel x86 architecture is in most cases downwardly compatible; therefore existing real–time software can easily be ported to newer hardware.

If a multiprocessor system is used for hard real–time tasks, the worst case execution time (WCET) of software is affected in several ways: The optimization techniques being used by a certain processor result in varying execution times. These are for example the use of caches and the branch prediction.

The interdependencies of the CPUs and the chip set have to be taken into account, too. If for instance there are two CPUs accessing the main memory at the same time on a SMP (symmetric multi processing) system, one CPU may have to wait for the other. If a processor has to perform I/O, its execution is delayed until it is granted access to the bus.

In this paper we present a software architecture for multiprocessor systems that allows the execution of a general purpose operating system (GPOS) and a real–time operating system (RTOS) in parallel on different CPUs. Thus it is possible to use all the comfort provided by the GPOS and to perform hard real–time tasks in parallel. This software architecture allows to control and to minimize response times and the WCET of real–time software. It is shown that a general purpose multiprocessor system can be configured in a way which leads to more deterministic computation times and faster response times compared to single processor systems. Our main objective is to make the computing power of modern general purpose multiprocessor systems available for real–time systems.

This paper is organized as follows: Section 2 gives an overview of current approaches for estimating the worst–case execution time of real–time software. Section 3 describes an existing method of running standard and real–time tasks in parallel which was the origin of our research work. This method is improved by our software architecture presented in section 4. To prove the advantages of our concept, measurement results are presented in section 5. The paper ends with a conclusion in section 6.

2. Current Approaches

The estimation of the best and worst case execution time can be done by modelling the hardware or by a measurement based approach. Colin and Puaut investigate in [2] the branch prediction unit of the Intel Pentium processor. In order to reduce complexity, they switched off the caches provided by the processor. Stappert and Altenbernd use a processor model including caches and pipelines in [7] and [8], but they simplified the analysis by assuming linear code only. A methodology
to analyze cache and pipeline behavior by abstract interpretation and pipeline modelling is presented in [4]. Here, only the inter-processor effects are investigated without taking the influences of the chip set into account. Petters describes in [6] a measurement based approach to estimate worst case execution times. A program is divided into basic blocks. The computation time of each block is measured on a given hardware. In [1] an extreme value statistics approach is described in order to deal with the measured worst case execution time. A large number of measurements is taken and an extreme value statistics density function is matched to the measured values.

In contrast to above-mentioned publications we try to adjust the real-time system in order to make its best and worst case execution times more deterministic. We take the whole architecture into account, regarding all the relevant aspects.

### 3. Underlying Concept

As we intend to provide a hard real-time system by adjusting the hardware and software of general purpose multiprocessor systems based on the Intel x86 architecture, we have chosen Linux with the real-time extension RTAI [3] as a basis of our work. The reason for this is the availability of the source code of both, Linux and RTAI, and the functionality already provided by RTAI: It has got real-time schedulers that support multiprocessors systems, and it can deal with semaphores, mailboxes and FIFOs.

RTAI itself uses the concept of running a GPOS as idle task of a real-time system. A tiny layer between the hardware and the GPOS is inserted (HAL), adding the real-time functionality to the existing operating system. All interrupts are intercepted by the RTAI kernel which decides if an appearing interrupt is relevant for real-time software or not. All non-relevant interrupts are forwarded to the GPOS being processed in the idle time only; the relevant interrupts are handled immediately. The GPOS itself is not allowed to enable or disable interrupts, this functionality is performed by the RTOS.

When using this architecture on a multiprocessor system as shown in figure 1, each CPU has got its own part of the RTAI kernel. On each processor, the GPOS is executed as idle task.

There are three major disadvantages when using the basic architecture of RTAI, regarding the influence of the WCET of real-time software:

- The GPOS is executed as an idle task on every processor. So if a real-time task is resumed on a certain CPU, the status of this processor is undefined. It cannot be said which cache lines are in cache and which have to be reloaded. The status of the Translation Lookaside Buffers (TLBs) and the branch prediction is not known, too. In addition, the time needed to switch from user mode tasks into the kernel mode has to be taken into account, as the privilege level of the processor has to be changed here.

**Figure 1: RTAI on a multiprocessor system**

- Each processor has to deal with irrelevant interrupts. Normally, there are only a few real-time devices and a lot of devices being used by the GPOS exclusively. In most cases the generated interrupts are irrelevant for real-time usage, but the RTOS has to react to all of them interrupting the execution of real-time tasks. If for example a key is pressed on the keyboard, an interrupt is generated which disrupts the execution of a real-time task.

- Each processor is able to perform I/O and is thus able to initiate DMA transfers. Previous studies have shown that the interaction between the chip set and the processors on a multiprocessor system cannot be neglected [5]. So if a real-time task wants to perform I/O, e.g. when writing to a hard disk, it may have to wait until the pending requests of other CPUs and peripheral devices are processed. This latency depends on the chip set and the communication protocols being used.

### 4. The RECOMS Software Architecture

In order to deal with the constraints mentioned in section 3, we have extended the architecture of the RTAI microkernel. Hardware and software of the real-time system are adjusted to get predictable best and worst case execution times. The influence of the GPOS on the real-time tasks is minimized allowing the utilisation of the GPOS and RTOS in parallel.

Our enhancements to RTAI are shown in figure 2. The CPUs are divided into two groups: One CPU is called the General Purpose Unit (GPU). On this processor the GPOS and its tasks are executed. The remaining CPUs are called the Real Time Units (RTUs). All real-time tasks are run on these processors exclusively. As the GPOS is bound to the GPU, the RTUs are available for real-time usage only. If a dual CPU computer is used, there is one GPU and one RTU, on a quad CPU system there are one GPU and three RTUs.

The interrupts are divided into two groups. The interrupts being irrelevant for real-time usage and the ones being trig-
The privilege level needs not to be changed. As the GPOS is executed on the GPU exclusively, the RTUs are always running in kernel mode. Thus there is no additional latency to interrupts caused by the switching from user mode to kernel mode.

The RECOMS software architecture leads to the following advantages for real-time software:

- The privilege level needs not to be changed. As the GPOS is executed on the GPU exclusively, the RTUs are always running in kernel mode. Thus there is no additional latency to interrupts caused by the switching from user mode to kernel mode.

- Real-time tasks are not interrupted by irrelevant demands. As only the interrupts needed for real-time tasks are routed to the RTUs, there is no additional delay to the real-time software.

- The caches and the TLBs are not displaced by the GPOS. As the GPOS with its tasks and interrupt handlers is executed on the GPU exclusively, the caches and the TLBs of the RTUs are always in a well known state. Thus it is possible to arrange the code of the real-time software a priori in order to minimize the displacement in caches and TLBs. If the relevant real-time code is very small, with this architecture it is now possible to lock this code in the cache of an RTU [10].

- There is no remaining interaction of the chip set and its communication protocols to real-time software. It is possible to postpone the I/O operations of the GPU which results in unhindered accesses of an RTU to a specific real-time device. If a real-time process wants to perform I/O, it is now able to access the chip set without interference of other non-real-time devices, whereby the real-time tasks have to synchronize their I/O requests by using semaphores.

5. Results

The advantages of our architecture are demonstrated by the central aspect of postponing the I/O accesses of the GPU. A real-time task is executed on the RTU of a dual CPU machine which performs a series of 1000 I/O accesses. This is needed if some data is written to a hard disk from a real-time task. Each measured value reflects the time needed to execute these 1000 I/O instructions. The code was fully arranged in the cache of the RTU and the interrupts were disabled during each measurement. In parallel to each measurement, the GPU accessed local and network file systems heavily.

Figure 3 shows the measured results for the situation when the GPU can perform its I/O accesses without any restrictions. The time needed to execute the 1000 I/O instructions varies from 195 microseconds to 940 microseconds. Compared to the best case, the execution time is increased by 382%.

If the I/O accesses of the GPU are postponed and the PCI devices are hindered from initiating DMA transfers during each measurement, we get the results as shown in figure 4. Now the measured values are only varying from 195 microseconds to 199 microseconds. Compared to the best case, the execution times are now increased by maximal 2%.

6. Conclusion

In this paper we present a software architecture for multi-processor systems based on the Intel x86 architecture that allows the execution of standard and real-time tasks in parallel. Hardware and software of the real-time system are configured in a way that leads to more deterministic computation times.

As the interdependencies of the CPUs and the chip set have a major influence on the execution time of real-time software,
these impacts have to be prevented: Standard and real–time tasks are executed on different CPUs, interrupts are routed to the CPUs executing the according tasks. I/O accesses of the GPOS are supervised by the RECOMS Software Architecture and can be postponed. Therefore it is now possible for real–time tasks to perform their accesses to peripherals without competing requests.

Compared to single processor systems the computation times of real–time software are more deterministic. There are no displacements in the caches due to the execution of the GPOS. Thus the caches of the RTUs can always be held in a well known state. Response times get shorter as the privilege level needs not to be changed if there is a real–time request. Real–time handlers are executed immediately and can be locked in cache leading to faster response times.

Future work deals with the synchronization issues of real–time tasks when performing I/O and initiating DMA transfers.

In particular, the I/O accesses of the RTUs should not occur simultaneously. Furthermore, the advantages when using a NUMA architecture as real–time system ought to be examined.

References


