RTEMS SMP FOR LEON3/LEON4 MULTI-PROCESSOR DEVICES

Daniel Cederman[1], Daniel Hellström[1], Joel Sherrill[2], Gedare Bloom[3], Mathieu Patte[4], and Marco Zulianello[5]

[1]*Aeroflex Gaisler, Kungsgatan 12, SE-411 91, Göteborg, Sweden, Tel: +46 31 7758650, Fax: +46 31 421407*
{daniel.cederman, daniel.hellstrom}@gaisler.com
[2]*OAR Corporation, 7047 Old Madison Pike Suite 320, Huntsville AL 35806, Tel: +1 256-319-2768, joel.sherrill@oarcorp.com*
[3]*Dept. of Computer Science, George Washington University, Washington, DC, gedare@gwu.edu*
[4]*Airbus Defence and Space / Space Systems, 31 rue des Cosmonautes Z.I. du Palays, 31402 Toulouse Cedex 4 France, Tel: +33562199032, mathieu.patte@astrium.eads.net*
[5]*ESTEC, Keplerlaan 1, PO Box 299, NL-2200 AG Noordwijk, The Netherlands, Tel: +31 71 565 8933, Fax: +31 71 565 5420, marco.zulianello@esa.int*

1. ABSTRACT

When multi-core processors are used in the space industry, they are mostly used in AMP configurations. The cost of increased complexity and difficulty in analyzing SMP systems has been deemed too high in comparison with the benefits of more processing power. A reason for this is the lack of easy to analyze operating systems capable of SMP configurations.

In this paper we present an European Space Agency (ESA) activity aimed at bringing easily accessible SMP support to GR712RC and ESA's future Next Generation Microprocessor (NGMP). This will be achieved by extending the RTEMS operating system with SMP capabilities and by providing parallel programming models and related libraries to exploit the intrinsic parallelism of space applications. The work will be validated by porting the single-core Gaia Video Processing Unit space application used in ESA's Gaia satellite project to RTEMS SMP running on GR712RC and NGMP.

The paper describes the ongoing effort and gives an overview of the challenges faced in extending a real-time OS to the SMP domain. The activity is funded by ESA under contract 4000108560/13/NL/JK. Gedare Bloom is supported in part by NSF CNS-0934725.

2. INTRODUCTION

Multi-core processors are commonplace in many areas, but currently not within the space industry. The added complexity in analysing software behaviour has so far not outweighed the benefit of fewer components and more processing power. This might however change in the future. To be prepared ESA commissioned the System Impact of Distributed Multicore System project [10] to better understand how multicore processors are best utilizied given the needs in the space industry. Conclusions drawn from the project were that two components were missing for optimal usage of ESAs multi-core NGMP for space payloads. The first was the lack of readily available and easily analysed real-time OS support for SMP configurations. The second was supporting libraries for parallel programming.

The activity presented in this paper takes a step towards remedying this situation by enabling and extending the SMP support of the real-time operating system RTEMS [8]. At the end of the activity it will be possible to develop SMP applications for RTEMS executing on both the commercially available GR712RC and on ESA's future NGMP multi-core processors. In addition the activity will provide implementations of APIs for efficient message passing and task-based programming in RTEMS. The SMP support and the task-based programming API will be validated by porting the Gaia Video Processing Unit from ESA's billion-star surveyor Gaia that was launched in December 2013 [5].

The NGMP system used will be the functional prototype GR-CPCI-LEON4-N2X which is a system-on-chip with a quad-core 32-bit LEON4 SPARC V8 processor connected to a shared 256 KiB Level-2 cache and several high-speed interfaces [3]. The GR712RC is an existing flight part with a dual-core LEON3FT SPARC V8 processor system suitable for advanced high reliability space avionics [1].

The activity is being carried out by a consortium led by Aeroflex Gaisler, developers of the LEON processor family, together with the OAR Corporation, maintainers of the RTEMS project, and Airbus Defence and Space, prime contractor of the Gaia software architecture.

3. RTEMS

RTEMS is a real-time operating system, popular in the European space industry. It is open-source with a com-

This paper was presented at DASIA 2014 in Warsaw.

munity driven development. Currently it can be used in a single software instance or in an AMP configuration where multiple single-processor instances run in parallel on different processors. Some support for SMP exists in the development branch of RTEMS, but no official release of the code has been made. The RTEMS SMP wiki gives an overview of the state of the SMP support [9]. Recently the support has been improved for ARM, X86 and PowerPC QorIQ by the community. At the start of the activity, however, the SMP code was not operational on SPARC platforms, such as the NGMP, due to missing implementations and/or bugs. The first part of the work thus consisted in identifying missing and broken functionality.

With an application written for a SMP system it is possible to achieve higher performance, but this comes at the cost of added complexity. Assumptions that are commonly made on single-core systems will be broken when executed on a multi-core platform. It is no longer true that the interrupt handler or highest priority task can be assumed to run in solitude, as a low priority task might run concurrently on another core. To ease the transition over to SMP, the scheduler should have support for specifying how tasks are allowed to be scheduled. RTEMS currently supports two basic SMP scheduler, the Simple SMP Priority Scheduler and the Deterministic Priority SMP Scheduler. They both have linear complexity and always schedule the P ready tasks with the highest priority, where P is the number of processing units. Currently they have no support for declaring processor affinity for tasks. The addition of this functionality will be one of the main contributions of this activity and is required for the proper implementation of the activity demonstrator.

The following subsections will go into more detail on the importance of processor affinity and give an overview of some of the other main areas of improvement addressed in this activity.

3.1. Processor Affinity

The processor affinity capability may be used in applications and the kernel in many different ways. Below is a list of a few common use cases in applications.

- For porting. Processor affinity may allow uni-core code to run by forcing sensitive code onto one specific processor, mimicking a uni-core system in parts of the application.
- For performance improvements. Always executing a set of non-interfering tasks on the same CPU increases the L1-cache hit-rate for that task set. It allows the user a more fine grained control over task scheduling and synchronization. The user may also do load balancing of the processors.
- Porting drivers. Uni-processor drivers typically turn off processor interrupt to lock out the interrupt service routine (ISR). However on a multicore system an ISR may be called on another CPU leaving the interrupt lock-out ineffective. By routing the interrupt to the same CPU as the user task calling the driver the problem just described is effectively avoided.

The processor affinity is also a key functionality for implementing many parallelization libraries, such as the ones described in this paper. Given the above use cases, one can easily recognize the great importance of the processor affinity capability for multi-core applications, and space applications are no different.

As part of the activity, the RTEMS SMP schedulers will be enhanced to support processor affinity. This is not trivial, as real-time scheduling with affinity remains an area of open research, especially for schedulability analysis of tasks that can migrate across cores [6]. When finished the schedulers will be able to limit the execution of a task to one or a set of processors, identified by the user through a new API. The intention is for the new affinity API to align with the APIs on GNU/Linux, including cpuset manipulation methods. Similar methods will be added to both the POSIX and the classic RTEMS API. The pluggable scheduler framework used by all schedulers in RTEMS to schedule tasks is not expected to change with the introduction of the affinity feature. The design is such that both non-affinity and affinity capable SMP schedulers are able to coexist in source. This ensures that affinity support will not be limited to the SMP schedulers, nor that it will impose the need for other schedulers to implement processor affinity.

The scheduler simulator provided with RTEMS will be used to simplify the implementation and verification work [4]. It allows the new capabilities to be tested prior to running on the real system, avoiding the full complexity of the multi-core system.

3.2. Cache Control

The RTEMS subsystem for cache control does currently not support SMP. In some cases it is only the local cores cache that is flushed, when all cores' caches should have been invalidated/flushed. This will be corrected by enhancing the cache control library to implement functions that operate on all cores cache. The planned approach is to signal to other cores using Inter Processor Interrupts (IPI) and implement an IPI handler that operates on the interrupted cores' cache.

3.3. Trace Library

RTEMS support a capture engine with tracing capabilities that can be used for debugging and to analyze behaviour and timing of the application. It is however currently limited to uni-processor support. Extending it to trace simultaneously on multiple processor cores imposes new problems. It would not be adequate that a processor hangs for other than a very short moment when adding a trace sample. One important reason other than it affects the application and determinism, is that tracing of interrupt and trap handlers are to be supported which are very

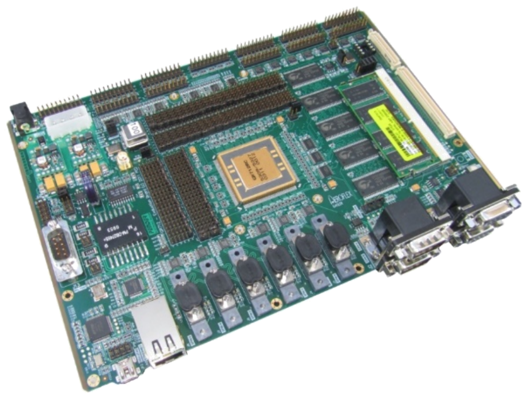

Figure 1. GR712RC Development Board

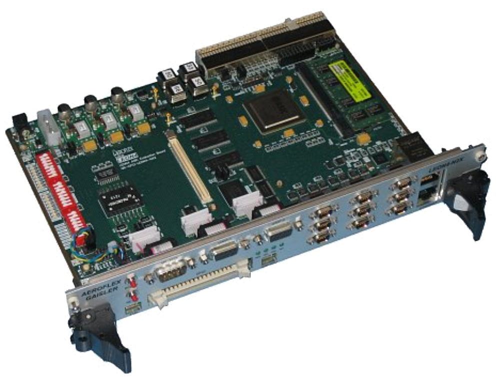

Figure 2. Quad-Core LEON4 Next Generation Microprocessor Evaluation Board

time critical. The planned SMP extension of the capture engine will therefore add trace samples using a lock-less implementation, which is supported by the LEON architecture. The implementation will revert to a global lock on unsupported architectures.

3.4. Porting to GR712RC/NGMP

The state of the RTEMS SMP on LEON when starting this work was broken. The early development has focused on repairing existing problems related to SMP on the targeted hardware and to add basic functionality required for the existing SMP implementation. To simplify development and debugging, the first stage of the implementation was performed using the GRSIM multi-core simulator [2]. It was then followed by verification on the actual GR712RC and NGMP platforms.

Both GR712RC and NGMP are supported by the same RTEMS BSP, the LEON3 BSP, which can be used for both LEON3 and LEON4. Thanks to the AMBA Plug&Play bus infrastructure and the LEON architecture as a whole, it was possible to maintain a minimum set of difference in code. Most of the platform specific issues are handled at boot time and does thus not affect RTEMS.

The NGMP has several new features that primarily facilitates easier AMP operation. This does not negatively impact SMP operation and the GR712RC and NGMP are fully compatible for areas such as interrupt handling and the basic function of peripherals. All RTEMS SMP development has been performed so that the code will be able to run on the hardware features provided by GR712RC. Backward compatibility with the new design ensures that the same code will run on the NGMP.

The Atomic Layer of RTEMS can now also take advantage of the CAS instruction available in LEON and SPARC V9 architecture.

4. PARALLEL PROGRAMMING MODELS

To efficiently take advantage of multi-core processors in SMP system requires different programming models than for the sequential case. Common models for parallel programming include task-based programming, where the work is divided into tasks that can be distributed among the available cores, and message passing, where threads on different cores communicates and synchronizes at given points using small efficient messages. Both types of programming models will be implemented as part of the activity. The support will be given in the form of two APIs adhering to the Multicore Task Management API (MTAPI) Specification [12] and the Multicore Communications API (MCAPI) Specification [11] by The Multicore Association, Inc. Both APIs have been designed with embedded systems in mind and tries to minimize latency and footprint at the cost of some of the flexibility of larger parallel programming suites.

The API specifications defines two concepts, *domain* and *node*. A node is an executable unit, such as a thread, a core or a cpu. A domain is a set of such nodes. It is up to the specific implementation of the API to decide the scope of the node and the domain. For the RTEMS implementation it is beneficial to define a node as an RTEMS task with affinity to one unique core. A domain in turn consist of a single multi-core processor. For the GR712RC this would mean one domain and two nodes, while for the NGMP it would mean one domain and four nodes. The two APIs will both use the RTEMS classic API for internal OS-specific operations and will utilize a bounded, pre-allocated, amount of memory and resources.

4.1. Message Passing (MCAPI)

The message passing API supports three types of communication. 1) Connection-less messages, 2) packet channels for variable size messages ordered in FIFO order, and 3), scalar channels, used to transfer words up to 64 bits in width in FIFO order. All three types allow the communication to be either *blocking* or *non-blocking*. In blocking mode the send and receive functions do not return until the message has been received or sent by the remote node. In non-blocking mode, the send and receive functions returns immediately. This allows for other work to be performed while waiting for the remote node. When no more work is suitable for processing until the message has been handled, the node can use a *wait* function that pauses execution until the remote node has sent or received the message.

There currently exists an open source implementation of MCAPI with a BSD license called OpenMCAPI, written and maintained by Mentor Graphics [7]. OpenMCAPI has been designed to be relatively easy to port, which lead it naturally to be the base of the RTEMS port. By not diverting too much from the original implementation, the extensive unit and functionality tests for OpenMCAPI can be reused. The main porting work will concist in providing RTEMS specific functionality for shared memory communication and to do adaptions to the code to cope with the new node definition that allows for nodes to share the same address space. The new implementation will also be optimized to assure that it performs well on the LEON architecture.

4.2. Task-based Programming (MTAPI)

The MTAPI library provides common functionality for task-based programming, such as spawning and waiting for tasks, but differs from other systems in the way it divides the conventional task into three different concepts. These are *job*, *action*, and *task*. A job represents some work that can be performed in the system, such as an FFT operation. An action is a concrete function that can perform the work represented by a job. For the example this could be a software function or a hardware implementation of the FFT algorithm. A task in the MTAPI setting is then an invocation of a particular job to be performed by an action. An action is node specific and a job could be associated with many actions. This allows for nodes with special hardware support to implement a job differently than a node that lacks it. It also allows for affinity and load distribution by selecting which nodes that will implement actions for a specific job. It is then up to the runtime to select which action that implements a given job that should be used for performing a task. The API also supports FIFO ordering and grouping of spawned tasks using queues and task groups.

There currently does not exist any public open source implementation of MTAPI. For this reason the MTAPI will implemented from scratch for RTEMS. The first version of the library will use a centralized scheme for load balancing, where nodes ask a common data structure for new applicable tasks when they have finished their previous work. This can later be extended with support for work-stealing for multi-domain settings, where communication and synchronization costs are higher. MTAPI could potentially take advantage of MCAPI as its internal communication protocol, but to lower the overhead and footprint, it was decided not to. Most applications will with high probability only utilize one of the APIs. MCAPI could however be a candidate for communication between domains.

5. DEMONSTRATOR

RTEMS SMP and the MTAPI task programming library will be evaluated by porting a flight-tested space payload software, the Gaia Video Processing Unit application. The porting will be performed by Airbus Defence and Space, which was the prime contractor for the development of the software architecture for the Gaia satellite. The Gaia VPU application is a typical example of one of the two use cases for multi-core processors identified in the SIDMS study. It is an application that requires both high processing performance and on-board re-programmability. The source code base used for the demonstration will be the same as the one used on the actual spacecraft and consists of about 32000 lines of code.

The VPU software is structured as a large software pipeline made of several relatively independent tasks. It is therefore a good candidate for parallelization. The source code base that will be used for the demonstrator is fully instrumented: the execution timings of all the tasks can be measured and reported. This will enable us to precisely monitor the performance of the parallel implementation and to optimize the selected architecture.

Airbus Defence and Space has access to the validation models of the VPU algorithms, and will therefore be able to validate the functional behaviour of the demonstration application using the same validation models as the one used for the actual spacecraft. In the actual GAIA spacecraft, the VPU software is executed on a PowerPC processor mounted on a Maxwell SCS750 board, which is ITAR. As the NGMP and other future LEON multi-core processors aim at becoming the European counterpart to the SCS750 board, it will be very interesting to compare the performances achievable by the demonstrator to the ones of the SCS750 board.

6. CONCLUSION

Increasing the usage of SMP systems in the space industry requires that well known and well understood software, such as RTEMS, is enhanced to fully support the SMP setting. The intent of this activity is to take RTEMS closer to this goal. At this stage of the activity it is already possible to execute software on GR712RC and NGMP in a SMP configuration using RTEMS. With the upcoming processor affinity support the porting of single-core or AMP software will be simplified and it will be possible to support high performance communication and task management APIs such MCAPI and MTAPI. The final porting of flight-tested space payload software from single-core to multi-core SMP will then act as a demonstration of the benefit and feasability of SMP for space applications.

REFERENCES

[1] Aeroflex Gaisler AB. GR712RC Dual-Core LEON3FT SPARC V8 Processor. www.gaisler.com/index.php/products/components/gr712rc. Accessed: 2014-01-10.

[2] Aeroflex Gaisler AB. GRSIM LEON MP Simulator. www.gaisler.com/index.php/products/simulators/grsim. Accessed: 2014-01-10.

[3] Aeroflex Gaisler AB. Quad-Core LEON4 Next Generation Microprocessor Evaluation Board. www.gaisler.com/index.php/products/boards/gr-cpci-leon4-n2x. Accessed: 2014-01-10.

[4] Gedare Bloom and Joel Sherrill. Scheduling and Thread Management with RTEMS. In *the 3rd Embedded Operating Systems Workshop (EWiLi)*, 2013.

[5] European Space Agency. ESA Science & Technology: Gaia. sci.esa.int/gaia/. Accessed: 2014-01-10.

[6] A. Gujarati, F. Cerqueira, and B.B. Brandenburg. Schedulability analysis of the linux push and pull scheduler with arbitrary processor affinities. In *25th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 69–79, 2013.

[7] Mentor Graphics. OpenMCAPI. bitbucket.org/hollisb/openmcapi. Accessed: 2014-01-10.

[8] OAR Corporation. RTEMS Real Time Operating System. www.rtems.org. Accessed: 2014-01-10.

[9] OAR Corporation. RTEMS SMP Wiki. wiki.rtems.org/wiki/index.php/SMP. Accessed: 2014-01-10.

[10] Mathieu Patte, Alfons Crespo, Marco Zulianello, Vincent Lefftz, Miguel Masmano, and Javier Coronel. System Impact of Distributed Multicore Systems. In *Data Systems In Aerospace (DASIA)*, 2012.

[11] The Multicore Association. Multicore Communications API Working Group. www.multicore-association.org/workgroup/mcapi.php. Accessed: 2014-01-10.

[12] The Multicore Association. Multicore Task Management Working Group. www.multicore-association.org/workgroup/mtapi.php. Accessed: 2014-01-10.