RTEMS SMP Executive Summary

Development Environment for Future Leon Multi-core

Contract: 4000108560/13/NL/JK

1 INTRODUCTION

1.1 Scope of the Document

This document establish the Executive Summary report for the RTEMS SMP activity. This development centered around RTEMS SMP is part of an activity initiated by the European Space Agency under ESTEC contract 4000108560/13/NL/JK.

The work has been performed by Aeroflex Gaisler AB, Sweden and ASTRIUM SAS, France and On-line Application Research (OAR) Corporation, USA. Since the start of the activity Aeroflex Gaisler has changed name to Cobham Gaisler and ASTRIUM has changed name to Airbus Defence & Space.

1.2 Reference Documents

[GR712]	"GR712RC Dual-Core LEON3-FT Development board", 2015-03-18, http://www.gaisler.com/index.php/products/boards/gr712rc-board
[N2X]	"GR-CPCI-LEON4-N2X – NGMP functional prototype development board", 2015-03-18, http://www.gaisler.com/index.php/products/boards/gr-cpci-leon4-n2x
[GR740]	"GR740: The ESA Next Generation Microprocessor (NGMP)", 2015-03-18, http://microelectronics.esa.int/ngmp/
[GRSIM]	"GRSIM LEON MP simulator", 2015-03-18, http://www.gaisler.com/index.php/products/simulators/grsim
[GAIA]	"Gaia overview", 2015-03-18, http://www.esa.int/Our_Activities/Space_Science/Gaia_overview
[RTEMS]	"RTEMS", 2015-03-18, http://www.rtems.org/
[SIDMS]	"SYSTEM IMPACT OF DISTRIBUTED MULTI CORE SYSTEMS (SIDMS) Final Report", 2011-11-22, http://microelectronics.esa.int/ngmp/SIDMS_Final_Report.pdf
[MCA]	"The Multicore Association", 2015-03-18, http://www.multicore-association.org/
[MCAPI]	"Multicore Association – MCAPI", 2015-03-18, http://www.multicore-association.org/workgroup/mcapi.php
[MTAPI]	"Multicore Association – MTAPI", 2015-03-18, http://www.multicore-association.org/workgroup/mtapi.php
[CTF]	"Common Trace Format (CTF)", 2015-03-18, http://www.efficios.com/ctf
[BT]	"Babeltrace", 2015-03-18, http://www.efficios.com/babeltrace
[LTTNG]	"Linux Tools Project – LTTng Integration", 2015-03-18, http://eclipse.org/linuxtools/projectPages/lttng/

1.3 Introduction to the activity

As multi-core processor devices have become more and more popular in the Real-Time Embedded Systems the space sector is bound to follow this trend too. ESA has launched several studies within the multi-core subject to analyse and explore possibilities and problems adapting new multi-core processor architectures. ESA is also active in developing future multi-core processors for space. The Next Generation Microprocessor (NGMP) activity will result in the GR740 ASIC, a quad-core LEON4 with a new bus architecture featuring for example a L2 cache.

The open source RTEMS Operating System is used in many European space missions. Several users in the space industry operates RTEMS in single-core configuration both on a ERC32 or LEON platform. The RTEMS community has started developing multi-core SMP support for RTEMS, it was started some years ago.

Studies have pointed out several problems in the existing multi-core software support. The objective of this activity is to improve the situation. Below is a brief summary of the objectives.

- Extend the RTEMS OS to support SMP configurations of the NGMP and GR712RC multi-core architectures. All changes to RTEMS are to be submitted to the RTEMS Community of acceptance into the main development stream
- Identify and port parallelisation libraries to exploit the parallelism of space applications
- Demonstrate the SMP support and programming model by parallelising a space application on multi-core LEON and compare the performance.

2 WORK PERFORMED

The following sub sections describe the major work performed within the activity. The first part was focused on developing the software support for RTEMS SMP and toolchain, whereas the last two parts can also be seen as a system test and performance study.

2.1 RTEMS SMP

The following sub sections described the major development work performed related to RTEMS SMP and the RTEMS toolchain.

2.1.1 Initial RTEMS analysis and getting existing RTEMS SMP working on LEON

During the first months of the activity the existing SMP support in RTEMS LEON BSPs were analysed. Before the activity started problems had been identified related to the LEON BSP initialisation and SMP. Several LEON BSP initialization and SMP issues were fixed and most SMP tests in the RTEMS test-suite was successfully executed. During the boot analysis it was discovered that the platform independent start-up code had some problems, cache inconsistencies could be seen and the test results of the RTEMS test-suite was in some cases not reliable. The discoveries were feedback to community and the specification.

2.1.2 Processor Set API

In order to have one common way to describe a set of processors within the RTEMS kernel it was proposed to define and develop a Processor Set API. The API was defined and implemented at the start of the activity to make it available to other code as early as possible. Changes were made to NEWLIB and RTEMS. The interface is located in NEWLIB sources at *libc/sys/rtems/sys/cpuset.h* and installed in *rtems/score/cpuset.h* among the RTEMS header file. The *spcpuset01* tests was added to the RTEMS test-suite.

The Processor Set API was used in all the developed RTEMS code where appropriate.

2.1.3 SMP Scheduler extended with CPU affinity

The ability to control which CPU or set of CPUs a task is allowed to execute on is an important feature in a SMP Operating System. For application developers it may be vital to optimize cache accesses, controlling the task scheduling, avoiding locking between tasks, load balancing, porting existing single-core code to SMP etc. It is also a fundamental requirement for many parallelisation libraries.

Existing user APIs used to control a task's CPU affinity was analysed. The two Task user APIs in RTEMS, the RTEMS classic API and POSIX thread API, where extended with the ability to read and write the current CPU affinity settings of a task. The following CPU affinity directives in the POSIX thread API in *pthread.h* were activated for POSIX threads in RTEMS:

```
int pthread_attr_setaffinity_np(
    pthread_attr_t *__attr,
    size_t __cpusetsize,
    const cpu_set_t *__cpuset
);
int pthread_attr_getaffinity_np(
    const pthread_attr_t *__attr,
    size_t __cpusetsize,
    cpu_set_t *__cpuset
);
int pthread_setaffinity_np(
    pthread_t __id,
    size_t  __cpusetsize,
    const cpu_set_t *__cpuset
);
int pthread_getaffinity_np(
    const pthread_t __id,
    size_t __cpusetsize,
    cpu_set_t *__cpuset
);
```

```
int pthread_getattr_np(
    pthread_t __id,
    pthread_attr_t *__attr
);
```

The following functions were added to the classic RTEMS API, declared in *rtems/rtems/tasks.h*:

```
rtems_status_code rtems_task_get_affinity(
   rtems_id                 id,
   size_t                   cpusetsize,
   cpu_set_t               *cpuset
);
rtems_status_code rtems_task_set_affinity(
   rtems_id                 id,
   size_t                   cpusetsize,
   cpu_set_t               *cpuset
);
```

The RTEMS pluggable scheduler API was extended with CPU affinity support. It is an option for a scheduler implementation to make use or ignore the CPU affinity properties. The existing RTEMS *Deterministic Priority SMP Scheduler* was extended to support CPU affinity. The new scheduler is named the *Deterministic Priority Affinity SMP Scheduler*.

The RTEMS test-suite was extended with new tests for the new scheduler. The following tests were added: *smpaffinity01*, *smpschedaffinity01*, *smpschedaffinity02*, *smpschedaffinity03*, *smpschedaffinity04*, *smpschedaffinity05*, *smppsxaffinity01* and *smppsxaffinity02*. A test-suite verifying the scheduler algorithm was developed using the RTEMS Scheduler Simulator framework. The simulator was extended to support SMP and several scheduler scenarios was described to trigger specific task scheduling cases. The scheduler simulator could successfully be used to describe scenarios hard to trigger in real hardware.

2.1.4 RTEMS Atomic Layer and toolchain

The atomic layer in an operating system provides platform independent support for various atomic operations. It is often used to implement SMP primitives such as semaphores and to protect internal structures. It was identified as an important piece in the SMP support before the activity started. Adding LEON support to the atomic layer using the LEON3/4 CAS instruction was proposed. However before the activity kicked off the RTEMS project selected to rely on the C11 standard for platform independent atomic support. This required GCC-4.7 or later and that the GCC backend supports the required atomic operations. With GCC-4.8.2 and GCC-4.9 released in May 2014 CAS support was introduced in the LEON backend. The goal within the activity then became instead to review, test and update the atomic CAS support of the LEON backend used to implement atomic operations and locking primitives of RTEMS SMP.

The GCC test-suite was interfaced to the LEON environment using GRMON2 Tcl scripts to support automated execution on GR712RC and LEON4-N2X targets and collecting execution results. Running the GCC test-suite is as usual a very time consuming effort but paid off. In the same context of LEON backend atomic support the machine instructions generated and the execution of parts of RTEMS SMP were analysed. Issues were found the GCC LEON backend and in the RTEMS spin-lock implementation. The issues found were propagated to the RTEMS community and patches were submitted to the GCC mainline repository.

To gain an overall bigger trust in the GCC-4.9.2 compiler the compiler was tested in an Linux-3.10 environment. Linux was selected due to its huge software support and our previous experiences. Various common benchmarks and test-suites were compiled and executed on the LEON4-N2X in SMP configuration. The performance figures showed an overall minor improvement since GCC-4.4.6, and in two specific cases a performance drop. Performance drops were analysed. At first it was believed to be a GCC SPARC backend issue which could signal a significant problem. Later it was found to be platform independent introduced around GCC-4.6 and of minor importance. The GCC maintainers decided to fix the problem in the mainline GCC rather than back porting the fix to the GCC-4.9 branch. The core-utils test-suite found a LEON4-N2X hardware MMU problem which was fed back to the NGMP CPU developer team and fixed for the GR740. Overall the GCC-4.9.2 compiler for LEON was found very stable.

2.1.5 Capture Engine

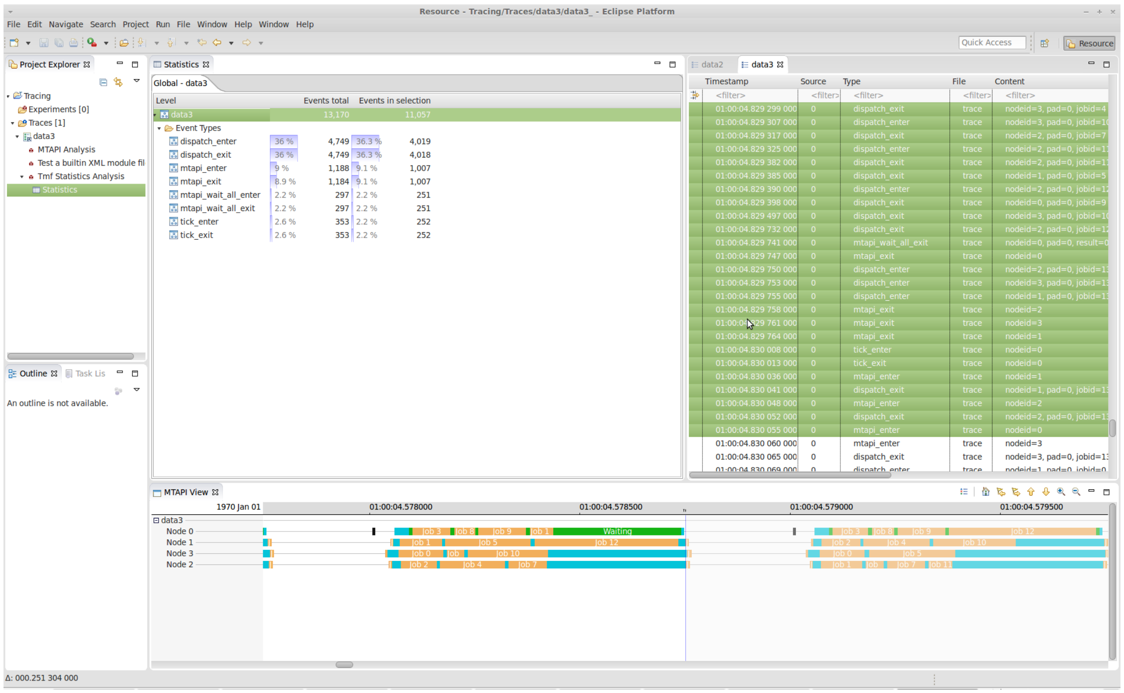
In order to overview and analyse events and in a complex SMP environment a Trace Library was to be developed. The existing RTEMS Capture Engine tracing solution was analysed and discussed within the community. The Capture Engine was to a great extent redesigned to support a number of new features, SMP support and to ensure that the trace recording had a minimal effect on the running SMP application.

The starting point of the project was that the existing Capture Engine was merely undocumented, no tests were present and SMP support fundamentally not working by design. It was concluded by the community that having two trace solutions in the RTEMS project was not optimal. Therefore the Capture Engine was redesigned inheriting most features already present. The community determined the path for the future RTEMS tracing capabilities which were explored during the activity. The work performed can be seen as an important step implementing the future plans and at the same time providing enough core functionality to allow it to be used today. The future plans involve inserting tracing functionality at link time into any function's entry and exit. It could be achieved automatically by extracting function prototype argument and return type information from a second source, for example the debug information or custom information. That information could serve as input for an automatically generated wrapper called at function entry/exit. The wrapper is responsible for doing logging and calling the original function to perform the expected functionality. Functions can be wrapped using the GNU LD linker's *–wrap=symbol* command line switch.

The focus were put on the core Capture Engine design itself and not so much the environment and wrapper generation. The following major pieces of the Capture Engine were developed:

- SMP support
- Support for variable sized record. This is a fundamental requirement to support tracing functions with different input argument types and count
- Support for command line interface were maintained and improved
- Real-time performance improvements were implemented by,
  - Redesign to removing dynamic memory in trace recording execution path
  - One capture engine per processor to minimise SMP locking
  - Capture record buffer read out partly redesigned
- RTEMS test-suite extended with new single-core and multi-core SMP tests: *capture01*, *smpcapture01* and *smpcapture02*

The new Capture Engine interface is declared in the *rtems/capture.h* header file.



Level		Events total		Events in selection
data3		13,170		11,057
Event Types				
dispatch_enter	36 %	4,749	36.3 %	4,019
dispatch_exit	36 %	4,749	36.3 %	4,018
mtapi_enter	9 %	1,188	9.1 %	1,007
mtapi_exit	8.9 %	1,184	9.1 %	1,007
mtapi_wait_all_enter	2.2 %	297	2.2 %	251
mtapi_wait_all_exit	2.2 %	297	2.2 %	251
tick_enter	2.6 %	353	2.2 %	252
tick_exit	2.6 %	353	2.2 %	252

Timestamp	Source	Type	File	Content
<filter>	<filter>	<filter>	<filter>	<filter>
01:00:04.829 299 000	0	dispatch_exit	trace	nodeid=3, pad=0, jobid=4
01:00:04.829 307 000	0	dispatch_enter	trace	nodeid=3, pad=0, jobid=10
01:00:04.829 317 000	0	dispatch_exit	trace	nodeid=2, pad=0, jobid=7
01:00:04.829 325 000	0	dispatch_enter	trace	nodeid=2, pad=0, jobid=11
01:00:04.829 382 000	0	dispatch_exit	trace	nodeid=2, pad=0, jobid=11
01:00:04.829 385 000	0	dispatch_exit	trace	nodeid=1, pad=0, jobid=5
01:00:04.829 390 000	0	dispatch_enter	trace	nodeid=2, pad=0, jobid=12
01:00:04.829 398 000	0	dispatch_exit	trace	nodeid=0, pad=0, jobid=9
01:00:04.829 497 000	0	dispatch_exit	trace	nodeid=3, pad=0, jobid=10
01:00:04.829 732 000	0	dispatch_exit	trace	nodeid=2, pad=0, jobid=12
01:00:04.829 741 000	0	mtapi_wait_all_exit	trace	nodeid=0, pad=0, result=0
01:00:04.829 747 000	0	mtapi_exit	trace	nodeid=0
01:00:04.829 750 000	0	dispatch_enter	trace	nodeid=2, pad=0, jobid=13
01:00:04.829 753 000	0	dispatch_enter	trace	nodeid=3, pad=0, jobid=13
01:00:04.829 755 000	0	dispatch_enter	trace	nodeid=1, pad=0, jobid=13
01:00:04.829 758 000	0	mtapi_exit	trace	nodeid=2
01:00:04.829 761 000	0	mtapi_exit	trace	nodeid=3
01:00:04.829 764 000	0	mtapi_exit	trace	nodeid=1
01:00:04.830 008 000	0	tick_enter	trace	nodeid=0
01:00:04.830 013 000	0	tick_exit	trace	nodeid=0
01:00:04.830 036 000	0	mtapi_enter	trace	nodeid=1
01:00:04.830 041 000	0	dispatch_exit	trace	nodeid=1, pad=0, jobid=13
01:00:04.830 048 000	0	mtapi_enter	trace	nodeid=2
01:00:04.830 052 000	0	dispatch_exit	trace	nodeid=2, pad=0, jobid=13
01:00:04.830 055 000	0	mtapi_enter	trace	nodeid=0
01:00:04.830 060 000	0	mtapi_enter	trace	nodeid=3
01:00:04.830 065 000	0	dispatch_exit	trace	nodeid=3, pad=0, jobid=13
01:00:04.830 069 000	0	dispatch_enter	trace	nodeid=1, pad=0, jobid=0

Illustration 1: LTTng showing RTEMS Capture Buffer of a LEON4-N2X MTAPI system

By making minor modifications to the LEON BSP interrupt handling code layout it was successfully demonstrated that also interrupt services routines can be traced.
A proof-of-concept SMP application was developed to test and demonstrate the usefulness of the Capture Engine on a SMP system. MTAPI task activities were traced by the Capture Engine and transferred over Ethernet NFS by a low-priority task. The resulting logs were described by meta data files according to the Common Trace Format (CTF) [CTF]. The trace could then be viewed and searched using the command line *babeltrace* [BT] tool or graphically using the LTTng [LTTNG] Eclipse plug-ins. A couple of LTTng plug-in Java classes were extended accordingly to describe the MTAPI objects and extract and display MTAPI specific data recorded. See the PC screen shot below captured from a LEON4-N2X board running MTAPI on RTEMS SMP.

2.1.6 Cache Manager SMP support

After the initial analysis of RTEMS and start-up boot code it was clear that cache inconsistencies could be experienced when one processor modifies instructions. Self modifying code typically is used during installing trap handlers. This occurs typically during boot when installing basic services, interrupt initialisation/registration and when the user installs custom error trap handlers.

The RTEMS Cache Manager was extended with SMP support and a test was added to the RTEMS SMP test-suite. The Cache Manager was extended to use Inter Processor Interrupts (IPI) to signal cache flush requests on another processor. A IPI interrupt handler was implemented to execute the request. Processors are described using the Processor Set API primitives.

2.1.7 LEON3 and NGMP BSPs

During the initial analysis and debugging effort of RTEMS start-up code and the LEON3 and NGMP BSPs the problems found were fixed and patches submitted. After that the boot was successful and most RTEMS tests from the SMP test-suite was working.

Driver wise the Interrupt controller, Timer and UART console drivers were of primary interest. They were examined, tested primarily using the RTEMS test-suite, SMP problems were fixed and drivers were extended to SMP where required support was missing. New per CPU interrupt controller macros were introduced in the BSP and support interrupt CPU affinity was added to the BSP. Interrupts can now be assigned to any processor at compile-time by the user. The user overrides a weak interrupt to CPU number table with a custom table.

Since all code developed were updated against the RTEMS mainline sources that changed often it was essential to build tests, execute tests, check results and create a test report in an automatic fashion, and regularly. The RTEMS test-suite was automated for the LEON environment using GRMON2 Tcl scripts. The test-suite was executed on both the targeted GR712RC and LEON4-N2X development boards. As found in the early analysis some of the tests in the test-suite could fail undetected when performing a standard check for start and end markers in the console output. To improve the situation it was decided to look at a test's exit code.

To allow the exit code to be propagated to the hardware debugger or bootloader the SMP shutdown mechanism was reviewed. Some issues apart from the exit code propagation was found. The SMP shutdown mechanism was partly redesigned both on a platform independent but mostly SPARC specific level. The test-suite execution result summary showed that about 10% of the tests return other exit codes than (5,0) which is the standard exit code. It made it hard to rely on the exit code to determine if a test failed or not. However it still served as a valuable input when manually verifying the execution results and failures were detected based on the exit code during the activity.

During the last part of the activity some optimizations were implemented to RTEMS SPARC register window overflow and underflow trap handling and SPARC interrupt level enabling and disabling trap handling.

2.1.8 LEON SMP development environment

In parallel to the RTEMS SMP development the SMP support of the LEON development environment was overviewed. Some of the major discoveries are listed below:

- Certain discoveries The new GCC-4.9 based toolchain required by RTEMS SMP generated DWARF-2 debugging information that was not fully compatible with older GDB version supported

by GRMON. The default debugging information format of GCC-4.9 is DWARF-3 which was not either supported by older GDB versions.
- GRMON did not support SMP or RTEMS-4.11 thread information
- GRMON issues when it came to restarting an SMP application and exit code

The ability to do C source level debugging is very important and considered a fundamental requirement. The solution to the above issues were developed and tested:

- GDB-7.7.x support in GRMON2
- SMP and bare metal thread model support in GRMON2
- RTEMS-4.11 and RTEMS-4.11 SMP task information support in GRMON2
- Found GRMON2 multi-core issues fixed and exit code can now be propagated to user shell

2.2 Parallelisation library

The following sub sections described the major development work performed related to the parallelisation library.

2.2.1 Parallelisation survey

The SIDMS study final results showed that the most interesting use case for multi-core processors within the space segment is scientific payload data processing applications. The survey focuses on the multi-core issues and needs of payload applications. To compare different available parallelisation techniques a set of criterion important for space applications in general and payload processing application specifically was identified based on earlier studies and experiences. Aspects taken into account are summarized below,

- the overall parallelisation scheme
- task synchronization method(s)
- task dispatching techniques supported
- support for hardware accelerators and multi-core systems
- compiler dependability and modifications required
- execution determinism
- software qualification
- adaptation complexity

A couple of widely used parallelisation libraries that were initially considered interesting given the context were selected for the analysis. The following libraries were selected and part of the analysis,

- OpenMP
- Intel Cilk plus
- Multicore Association (MCA) APIs [MCA]
- POSIX Threads

The libraries were scored based on the summary of all above described criterion. The results justified that the Multicore Association APIs was selected. The scores of MTAPI [MTAPI] were overall high apart from the adaptation complexity.

2.2.1.1 MCA MTAPI

The Multicore Association distributes the open specification MCAPI and MTAPI. The MCAPI [MCAPI] specification provides platform independent communication services based on message passing over any communication channel. Whereas the MTAPI specification provides task management services, some of which are common in all task-based programming libraries. This includes spawning tasks and waiting for tasks to be completed. One difference between MTAPI and other libraries, however, is in its support for heterogeneous platforms.

When spawning a task one does not provide a function pointer. This would not work when communicating between different types of hardware architecture. Instead an identifier is used that corresponds to some abstract work. In MTAPI this identifier is called a *job*. A processing node can register an *action* for a specific job. An action is a software or hardware function that the node can execute to perform the abstract work specified by the job. Spawning a task thus consists of creating a task object with input arguments for the task and a job identifier. The MTAPI scheduler then decides which node, out of the ones that have registered an action for the job, should perform the actual task, using its registered action. This system allows different nodes in the system to perform operations in the manner that is the most efficient for that hardware. It also allows for work sharing between different types of hardware, since there is no exchange of function pointers.

2.2.2 MCA Parallelisation Library implementation

MCAPI and MTAPI has been design such that a MTAPI implementation can rely on MCAPI for a platform independent and scalable communication service between processor cores or processor devices. Both specifications were analysed in parallel with the ongoing analysis of the demonstrator application requirements. At first it was determined only a minor number of MCAPI services was required, mainly to support MTAPI communication. During the development process it became evident that it was beneficial to make a smaller more efficient custom communication solution for the implementation of MTAPI. The communication in this case is limited to on-chip communication between processor cores using shared memory resources and services to generate interrupt for signalling.

The MTAPI implementation was written in C and performs no dynamic memory allocation. The maximum number of jobs, groups, active tasks, action and queues, are all specified at compile time. To lower latencies, all common functions, such as spawning and dispatching tasks, have been written in a lock-free manner. Less time critical operations, such as the creation and deletion of queues, are instead blocking. A *node* is defined as a RTEMS task with affinity to a unique core. The nodes communicate via shared data structures. A centralized scheme is used for load balancing, where nodes ask a common data structure for new applicable tasks when they have finished their previous work.

Most MTAPI services were developed and unit tested by implementing a test-suite supported by coverage analysis. The coverage results were close to full coverage at 92.5%. A system level test was developed to mimic the expected task scheduling behaviour of the demonstrator. It was developed in order to verify the MTAPI functionality in an RTEMS SMP LEON multi core environment prior to running the demonstrator. This turned out to be a successful approach saving a lot of time in the project.

Performance tests performed showed that creating a MTAPI group, starting 16 tasks on other CPU cores and waiting for the group execution completes took in average about 50us on a LEON4-N2X 150MHz. This results in an about 300KHz MTAPI task start and join rate. The tasks did not perform any work.

2.3 Demonstrator

The demonstrator consists of a flight-tested space payload software, the Gaia Video Processing Unit (VPU) application. The original software was designed to run on a single-core PowerPC processor mounted on a Maxwell SCS750 board (ITAR) using VxWorks. The original VPU software consists of about 32000 lines of code.

The major development work performed related to the Gaia VPU application demonstrator are summarised below.

- Ported to LEON architecture and RTEMS-4.10 single-core configuration
- Ported to RTEMS-4.11 SMP and parallelisation using MTAPI
- Analysed and after restructured software to optimize task scheduling to better utilize the processor cores present.
- Measured and compared performance in different hardware and software configurations

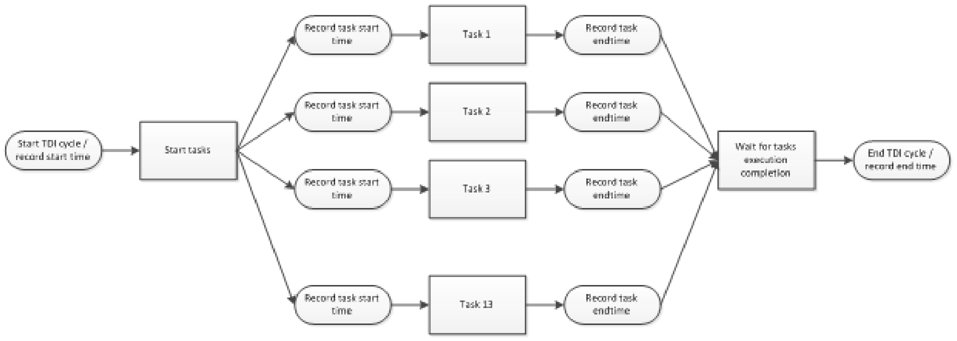


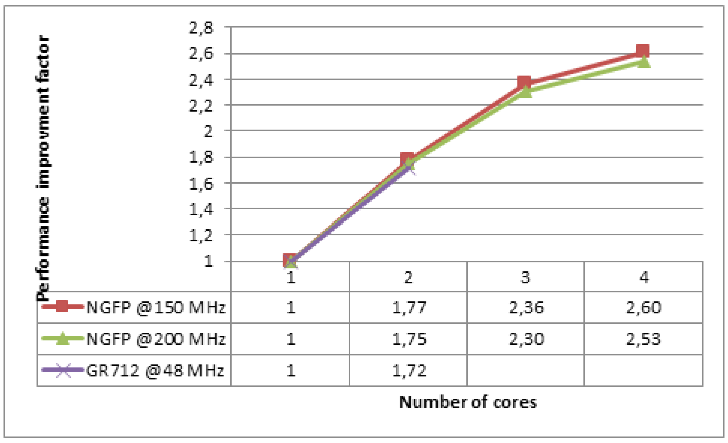
Illustration 2: Task execution flow graph

The VPU software is structured as a large software pipeline made of several relatively independent processing functions. To parallelise the application, these functions were initially grouped into 13 tasks that works on independent data sets and thus can be executed concurrently. Using MTAPI these tasks are all scheduled concurrently once per Time Delayed Integration (TDI) cycle. The cycle frequency is around 1KHz. The duration of each task was recorded for each TDI cycle using a high resolution timer.

The parallelisation of the application was found to be straightforward. Initially the performance was poor. To gain understanding on what was the limiting factor the actual time spent in the processing tasks were measured by recording the start and end time of tasks. How well the application was parallelised could this way be quantified and compared between different hardware and software configurations. It was concluded that processors were idling at certain times waiting on other tasks to complete their computations. The application code was restructured into 16 tasks. By splitting the longest executing tasks into smaller parts the work load could be spread more evenly between cores. The below table shows the hardware configurations for which the demonstrator was run.

Board	**CPU core clock speed**	**Memory type and speed**	**Memory timings**
LEON4-N2X	150 MHz	64-bit DDR SDRAM @ 300MHz	Col 10, ref 7.8us, trfc 82ns
LEON4-N2X	200 MHz	64-bit DDR SDRAM @ 300MHz	Col 10, ref 7.8us, trfc 82ns
GR712RC	48 MHz	32-bit SDR SDRAM @ 48MHz	Col 10, CAS 2, ref 7.8us
GR712RC	80 MHz	32-bit SDR SDRAM @ 80MHz	Col 10, CAS 2, ref 7.8us

Table 1: Test hardware configurations



	1	2	3	4
NGFP @150 MHz	1	1,77	2,36	2,60
NGFP @200 MHz	1	1,75	2,30	2,53
GR712 @48 MHz	1	1,72		

Illustration 3: Performance improvement factors with number of cores used (higher is better)

The LEON4-N2X at 150MHz achieved the highest performance factors. The performance scaled well up to two cores (1.77x) and three cores (2.36x), but gained little going up to four cores (2.60x). The GR712RC performance improvement utilising both cores was 1.72x. The dual-core performance of the GR712RC per MHz was about 13% lower compared to LEON4-N2X running at 150MHz.

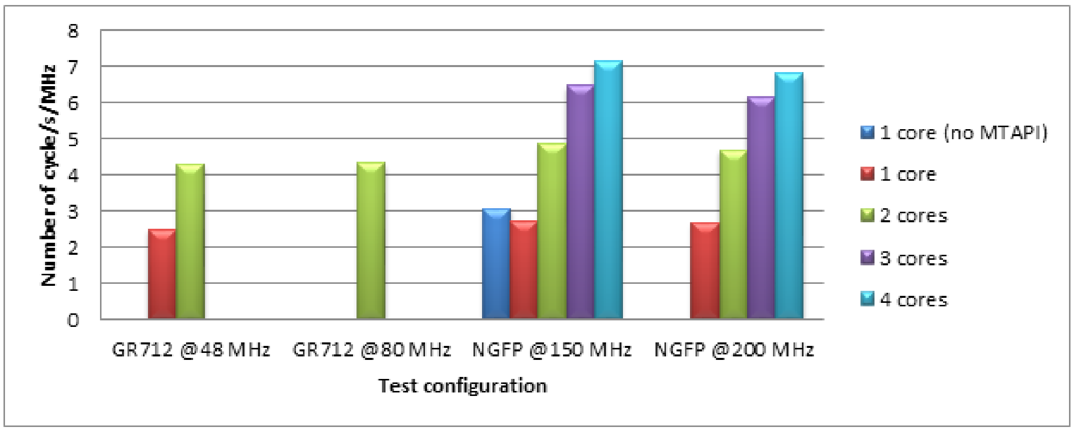


Illustration 4: Number of TDI cycles per second and per MHz (higher is better)

After normalizing the number of TDI cycles with the CPU frequency one can see how performance scales between the configurations.

The overhead caused by MTAPI+RTEMS was measured to about 12.5% of the overall execution time. The figure was calculated by comparing the application in a RTEMS-4.10 single-core configuration to a MTAPI parallelised application in a RTEMS-4.11 SMP single-core configuration.

The L2 cache hit ratio of the LEON4-N2X was measured using the L4STAT unit. GRMON periodically read the cache hit/miss/access statistical counters. The average L2 cache hit ratio was measured to 96.5%.

3 CONCLUSIONS

This activity has resulted in that the SMP support of RTEMS on the LEON3 and NGMP BSPs is working and usable in practise. The SMP scheduler and BSP interrupt handling now supports CPU affinity. It makes it easier to adopt RTEMS SMP and the fundamental pieces are in place for SMP applications. The new LEON toolchain for RTEMS has been extensively tested building a large confidence. The LEON development environment now supports RTEMS SMP.

All code developed for RTEMS Operating System, BSPs and toolchain has been submitted upsteams to respective project repository and is freely and publicly available. It will be part of the next stable RTEMS release RTEMS-4.11. This ensures that the software developed are available and possible for reuse in future ESA space missions.

Existing parallelisation techniques were analysed to find that the MTAPI task management library meets space payload application requirements best. The MTAPI specification was implemented for RTEMS SMP on LEON. The MTAPI library can be used to help existing and new space software to exploit multi-core processor resources using a platform-independent API.

The Gaia VPU application was successfully ported from the single-core PowerPC VxWorks environment to RTEMS SMP on the GR712RC and LEON4-N2X platforms. By using RTEMS SMP CPU Affinity Scheduler together with the MTAPI library the application could effectively be parallelised and partly restructured to improve the performance significantly. All the developed pieces were integrated and it was demonstrated that they worked together by verifying the correct execution and output. Various performance figures were measured in different configuration using the extensive performance instrumentation framework of the VPU Gaia test platform. Performance was compared between different hardware and software configurations.

The performance increase of the parallelised demonstrator scaled well up to three CPU cores. On a LEON4-N2X in a quad core configuration it resulted in a speed up of about 2.6 times the single-core performance. By measuring the subtask start and exit times it could be concluded that only 273% of the actual time was spent in the calculation algorithm. The other time spent is overhead and waiting for other CPUs to complete their work load. This leaves room for performance improvements, however could require a more fine grained parallelisation of the space application and that MTAPI might not be an optimal solution in that case. It shows that a payload application like the demonstrator can be parallelised using MTAPI in a couple of months to gain a significant performance improvement over the single-core system.

Based on the NGMP prototype performance results and assuming that the future GR740 runst at 250MHz using SDR SDRAM memory the GR740 would be fast enough to run the existing VPU application. Comparing against the reference PowerPC SCS750, Airbus Defence & Space concludes that the GR740 could be a future candidate to replace the PowerPC for this kind of application. Its notable that the power efficiency of the GR740 in this case is much better, approximated to consume only one third of the power required today.