Reference:
SPB-DE4FLMC-301-FR-001

Issue:
1 - 20/04/2015

Revision:
0 - 20/04/2015

Distribution Code:
Restricted Distribution

UNIVERSITÀ DEGLI STUDI DI PADOVA

SPACEBEL

embedded brains

TRP

Development Environment for Future LEON Multi-Core

RTEMS SMP Final Report

Prepared by:

Fernand Quartier

Verified by:

Fernand Quartier

Approved by:

Paul Parisis

Abstract

This document is the final report on the RTEMS SMP development environment development as executed by Spacebel, embedded brains and the University of Padua.

Keywords

Real Time Operating System, Multi-Core, Scheduling, Symmetric Multi-Processing

Contract

Contractual	:	✓
Contract issuer	:	ESA
Contract n°	:	400010108771/13/NL/JK

Classification

General Public	:	
Industry Programme	:	
Restricted Dispatching Programme	:	✓
Confidential Programme	:	

Configuration

Configured document	:	✓
Non-configured document	:	

References

Reference	:	SPB-DE4FLMC-301-FR-001
Issue	:	1 - 20/04/2015
Revision	:	0 - 20/04/2015
Number of Pages	:	35
Word Processor	:	MS Word

Internal Distribution

Name	Dept.	Copies	For Information	For Action	Comments
Fernand Quartier	Space	1		✓	For preparation
Fernand Quartier	Space	1		✓	For verification
Paul Parisis	Space	1		✓	For approval
V. Demeuse		1		✓	For configuration

External Distribution

Name	Dept.	Copies	For Information	For Action	Comments
Marco Zulianello	ESA	1		✓	For acceptation

Document Change Log

Issue	Issue Date	Pages Affected	Relevant Information
1.0	20/04/2015	All	Initialisation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose of the Document

This document is the final report on the RTEMS SMP development environment development as executed by Spacebel, embedded brains and the University of Padua.

1.2 Scope of the Document

This document covers the document item FR Final Report identified in the SoW [AD01].

1.3 Readership of the Document

This document is targeted at ESA Project Manager and Technical Officer in charge of the DE4FLMC project.

1.4 Organisation of the Document

This document is organised as follows:

- Chapter 1 is this introduction,
- Chapter 2 describes the main project objectives,
- Chapter 3 presents the state and results of the developments of RTEMS SMP and its tools,
- Chapter 4 takes a look of the state of the art of multi-core real-time systems,
- Chapter 5 describes briefly the findings on parallel libraries,
- Chapter 6 reports the results of the Proba demonstrator and the findings,
- And, chapter 5 presents the project conclusions.

1.5 Property of the Document

1.6 Applicability of the Document

This document applies to the developments made in the scope of the contract.

1.7 Applicable Documents

The following documents are applicable to the project. In the body of the text these documents are referenced as listed here below.

[AD01] Development Environment for Future Leon Multi-Core - Statement of Work
TEC-SWS/12-530/SoW - Issue 1.1 - 31.05.2012

[AD02] SMP Demonstrator - Architectural Design Document
SPB-DE4FLMC-301-ADD-001

1.8 Reference Documents

The following documents provide background reference. In the body of the text these documents are referenced as listed here below.

[RD01] RTEMS-SMP C User's Guide
On-Line Applications Research Corporation

1.9 Conventions

None.

1.10 Glossary

None.

1.11 Abbreviations and Acronyms

ADD	Architectural Design Document
ADxy	Applicable Document xy
DE4FLMC	Development Environment for Future LEON Multi-Core
DHS	Data Handling System
ESA	European Space Agency
GR	Gaisler Research
IO	Input/Output
NGMP	Next Generation Multi-Processor
OBC	On-Board Computer
OBSW	On Board Software
OS	Operating System
RDxy	Reference Document xy
RTOS	Real Time Operating System
SDE	Software Development Environment
SMP	Symmetric Multi-Processing
SoW	Statement of Work
SPB	Spacebel
SRD	System Requirement Document
SRS	Software Requirements Specification
SUM	Software User Manual
SW	Software
TBC	To Be Confirmed
TBD	To Be Defined
TBW	To Be Written
TC	Telecommand
TM	Telemetry

2 MAIN PROJECT OBJECTIVES

The main project objectives where to develop an RTEMS SMP run-time and development environment that allows to exploit multi-core processors using RTEMS SMP, investigate its usability for space projects and to explore parallel libraries.

Estec has decided to issue a parallel contract with Cobham (Aeroflex) Gaisler, OAR Corporation and Airbus Defence and Space on one side, Spacebel, embedded brains (EB) and University of Padua (UoP) on the other side.

The Spacebel consortium had put the accent on objectives that were more concentrated on real time behaviour, determinism and general usability in typical space applications, especially for satellite Data Handling Systems (DHS).

In the Spacebel consortium, there were several priorities, use cases and roles.

UoP was in the first place concerned about the fundamentals about scheduling, priority management, predictability and worst-case analysis from the academic perspective.

Embedded brains was primarily concerned about a solid foundation that allows for a best effort real-time system that exploit de available resources to a maximal extent. Embedded brains did all RTEMS related development work.

The primary concern of Spacebel was to analyse the usability of the NGMP processor for satellite DHS applications using the first core as main processor, and its potential to exploit the capacity of the remaining cores for additional payload or instrument controllers or data processors. Spacebel managed the project, did investigation on and implementation of the parallel libraries and build a representative Proba demonstrator.

3 RTEMS SMP

3.1 Starting Point at Project Begin

RTEMS is a real-time operating system with more than 20 years in operation which was never designed for SMP systems. The previous attempt to get it running on multi-core machines had some severe implementation flaws, e.g. usage of test-and-set locks (TAS). Large parts of the existing SMP support were rewritten from scratch during the project.

3.2 Strategy: Single-Core to Multi-Core

The strategy to get from a single-core operating system so something that reasonably well supports multi-core was like this.

- Evaluate high-level APIs. Select alternatives for APIs not available to or unsuitable for SMP systems, e.g. task variables, interrupt or pre-emption disable to ensure mutual exclusion. Remove these APIs or add run-time errors or assertions upon use. Avoid these APIs in the RTEMS code base and ensure that the test suite passes.
- Prepare the tool chain. This includes C11 (ISO/IEC 9899:2011) and C++11 (ISO/IEC 14882:2011) support for the LEON3/4 processor.
- Evaluate and choose low-level synchronization primitives.
- Get it running on multi-core with minimal effort.
- Add new APIs (e.g. partitioned/clustered scheduling).
- Add profiling to identify bottlenecks.
- Get rid of bottlenecks step by step.

3.3 Tool Chain

Some effort was necessary to get a suitable tool chain for the LEON3/4 processor. The Binutils had to support the compare-and-swap (CAS) instruction available for the LEON3/4 processor. The GCC had to support the LEON3/4 memory model and atomic operations. This was done in cooperation with the community maintaining Binutils and GCC (mainly AdaCore) and Cobham Gaisler. It is recommended to use Binutils 2.25 and GCC 4.9.3.

3.4 Low-Level Synchronization

The low-level synchronization primitives are implemented using C11 or C++11 atomic operations so no target specific hand written assembler code is necessary. The prime requirement for low-level mutual exclusion is FIFO fairness since we are interested in a predictable system and not maximum throughput. With this requirement the solution space is quite small. For simplicity the ticket lock algorithm was chosen. The API however is capable to support for example Mellor-Crummey Scott (MCS) locks. This may be interesting in the future for systems with a processor count in the rage of 32 or more. A barrier operation was implemented as a sense barrier. The

barrier is currently only used in the test suite. Some areas use lock-free synchronization. These are the lowest level thread dispatching and context switches, the thread lock (this lock may change in case a thread blocks on a resource) and thread priority updates.

3.5 Early Steps using the Giant Lock

It is fairly easy to get a single-core operating system running on multi-core. You simply have to encapsulate the operating system state and protect it with one global recursive lock - the Giant lock. One exception is the lowest level thread dispatching since this is an asynchronous operation on multi-core machines. A scheduler decision on the current processor must be carried out on other processors using an inter-processor interrupt. This is not an instantaneous procedure and thus you may have several thread dispatches in the air.

A system using a Giant lock however is useless on systems with more than two processors since it is the major bottleneck that limits the system performance drastically. A test case tmtests/tmfine01 was added to measure the performance of some simple use cases, e.g. obtain and release mutex, send an event, send a message.

Let's consider a simple test case with worker tasks each performing mutex obtain and release operations using a private mutex. With up to four active workers each worker has its own processor on the NGMP. There is a slight increase in the total operations from one to two processors, but it is not doubled. Adding more workers degrades the overall performance. This is due to the Giant lock for which all workers fight.

3.6 Profiling

To identify the bottlenecks in the system support for profiling of low-level synchronization was added. This enables to spend the testing budget efficiently and concentrate on the hot spots. The profiling is a build time configuration option and is implemented with an acceptable overhead even for production systems. A low-overhead measurement of short time intervals must be provided by the hardware. This turned out to be problematic on the NGMP prototype system. The profiling information available per-processor is

- the maximum thread dispatch disabled time,
- the mean thread dispatch disabled time,
- the total thread dispatch disabled time,
- the thread dispatch disabled count,
- the maximum interrupt delay (needs special hardware support),
- the maximum interrupt time,
- the mean interrupt time,
- the total interrupt time, and
- the interrupt count.

The profiling information available for SMP locks is

- maximum acquire time,
- maximum section time,
- mean acquire time,
- mean section time,
- total acquire time,
- total section time,
- usage count, and
- contention count for an initial queue length of 0, 1, 2 and more than 2.

Profiling reports are generated in XML using the test suite (more than 500 test programs). This gives a good sample set for statistics.

3.7 Partitioned/Clustered Scheduling

Clustered/partitioned scheduling helps to control the worst-case latencies in the system. The goal is to reduce the amount of shared state in the system and thus prevention of lock contention. Modern multi-processor systems tend to have several layers of data and instruction caches. With clustered/partitioned scheduling it is possible to honour the cache topology of a system and thus avoid expensive cache synchronization traffic. It is easy to implement. The problem is to provide synchronization primitives for inter-partition synchronization. In RTEMS there are currently three means available

- events,
- message queues, and
- the multi-processor resource sharing protocol (MrsP).

The MrsP is a generalization of the priority ceiling protocol. It uses a helping mechanism which allows threads to execute temporarily on a foreign partition in case they get pre-empted by a higher priority thread and are an owner of a MrsP semaphore with other threads waiting to get ownership. The helping mechanism is implemented in the scheduler and adds a considerable complexity to its implementation. An implementation with reasonable worst-case execution times is an open topic.

3.8 Fine Grained Locking

As a first step fine grained locking was implemented for events, semaphores and message queues. Fine grained locking means that each object has its own lock to protect the object state (e.g. the thread queue with blocked threads, if the mutex is available or not or the pending messages). In case the scheduler is necessary to carry out a block or unblock operation, then a handover to the scheduler lock must take place. The semaphores and message queues use thread queues which

are a building block for all objects which allow threads to block. So the basic work is done to implement fine grained locking for all synchronization objects provided by RTEMS. The results of the simple benchmark test tmtests/tmfine01 obtained on the NGMP system show that the fine grained locking implementation scales well with the count of active workers and outperforms the old implementation also in the one processor case significantly.

	Operations on one Processor (Old)	*Operations on one Processor (New)*	*Change*
Events to Self	44262	82150	86%
One Mutex per Worker	42520	67998	60%
Message to Self	32591	58596	80%

Table 1: Comparison of operations on one processor (50MHz NGMP)

3.9 Time Keeping

The nanoseconds extension used to get timestamps below the system tick resolution was broken by design on SMP. In addition the usage of an SMP lock to get the timestamps was a performance bottleneck. So a completely different implementation was necessary. After an evaluation of existing implementations the FreeBSD time counters were selected. They were ported to RTEMS and show excellent results.

A high performance timestamp implementation is vital for the overall system performance. During each thread dispatch some timing information is updated using the current uptime. It is also necessary for low overhead run-time tracing.

3.10 Open Source

RTEMS is available to everyone without registration or other obstacles (git clone git://git.rtems.org/rtems.git). There was some work done sponsored by other users during the ESA project cycle. They use it for large scale banknote processing, x-ray and particle detectors and high performance digital audio broadcast. The work includes the basic SMP scheduler framework, SMP support for ARM and PowerPC and the network stack port from FreeBSD 9.

What is the benefit for ESA? The ARM and PowerPC support helped to speed up development due to better debug support (e.g. Lauterbach PowerTrace), more stable targets and different timing conditions revealing more bugs. The network stack port from FreeBSD leveraged the prototype implementation for fine grained locking.

3.11 Status

RTEMS is partially ready for production systems if you know its limitations. It has a solid low-level implementation of the following components

- the low-level synchronization,

- thread migration and processor assignment,
- SMP scheduler framework,
- partitioned/clustered scheduling,
- thread queues (building block for objects which may block a thread), and
- thread-local storage.

It is a suitable as a low-overhead guest system for space and time partitioning and of course as a stand alone operating system.

3.12 Future Work

Everything is ready to eliminate the Giant lock entirely. This is more or less a simple but somewhat labour intensive task. Proper priority queues for partitioned/clustered scheduling (combination of FIFO and per scheduler priority queues) are missing. Support for priority boosting is desirable for simple priority inheritance mutexes. Implementation of the O(m) independence-preserving protocol (OMIP) is highly recommended since this is a generalization of the priority inheritance protocol to partitioned/clustered scheduling and can be used for a general purpose mutual exclusion primitive. Each scheduler should have its own lock to enable a scalable system. New APIs for objects without an identifier to object translation and workspace usage are necessary due to performance reasons. In applications with fine grained locking (e.g. the FreeBSD network stack) the identifier to object translation turned out to be an unacceptable overhead.

4 A LOOK INTO THE STATE OF THE ART BY UOP

4.1 Premises

The on-board software (OBSW) is traditionally designed for single-core processors. As the available processor platforms have been single-core up to very recently, there was no point in investigating how to generalize software architectures to make them more scalable.

The situation however is rapidly changing: the trend toward the adoption of multi-core processor platforms is evident in most application domains [3], even those that are close for needs and practices to the OBSW domain. The fact is then that traditional OBSW architectures, designed and implemented with the single-CPU assumption in mind, do not scale. The situation has revived old approaches that leverage the inherent parallelism of new-generation platforms (including GPUs) without the need to redesign the whole SW. Those approaches however are limited by the amount of parallelism that (perhaps hidden to the programmer, but visible to a compiler-level engine) actually exists in the current program, and by the amount of resources that can be afforded in adjusting the program code to make it more parallel. This approach goes counter the software engineering best practice of "enforcing intentions", which suggests that if I want scalability in the processing capacity of my program (that is to say, potential for parallel execution), I have to have it in my design, not seek it from a compiler regardless. Compilers help, of course, but cannot and should not take the placeof software designers. Indeed, using macros to define sections of code that can be run in parallel (while relying on the compiler for producing actual parallel code, as in e.g. [1]) or using external libraries to explicitly state when to create multiple or parallel instances of specific functions, as in e.g. [2], has evident limits of traceability to verifiable design intentions.

The immediate solution that comes to mind to resist the change of paradigm caused by the advent of multi-core processor platforms is to not parallelize the OBSW and rather assign it, monolithically, to one specific core, leaving the other cores to running possibly parallel payload SW. Evidently, this is a shortterm solution, which postulates that the computational need of future OBSW will not ever exceed the capacity of a single core. We know this is not a sound claim: we don't need to force more computational load into the OBSW but we can't certainly close the door to the opportunity of doing more with it.

Arguably therefore, it is opportune that clean and scalable solutions are devised for OBSW to make sound use of the new multi-core processor platforms, if not for the immediate present, for the (very) near future. The work conducted in this study aims at providing a possible valid solution to this problem in the specific context of a LEON4 processor architecture dressed with the RTEMS operating system. The question we addressed was how can we modify RTEMS so that it can adequately (for average performance and time predictability) manage a symmetric multiprocessor, SMP, such as the LEON4?

4.1.1 Dilemmas

A scalable solution, however, must have specific traits in order to be viable for use by the OBSW and the payload SW, without postulating their segregation (which is not obvious to achieve in a shared-memory multi-core processor with no support for memory management).

The envisioned solution should:

- exploit as much as possible the platform capacity. This theoretical issue relates to the choice of scheduling algorithm used to manage software execution on the system

- allow the application to have a predictable run-time behavior. This theoretical issue relates to the tools used to determine whether critical programs can be assured to execute within their assigned deadline

- provide ways for software assigned to different cores to communicate efficiently and predictably. This theoretical issue relates to the need of having a sound mechanism to synchronize parallel/concurrent software without incurring distributed and unbounded overhead

- be free from untenable performance penalties and overheads. This practical issues relates to the need of having a responsive system and it is addressed in the part of this report produced by embedded brains.

4.2 Scheduling algorithm

In the last few years, a vast body of literature has been produced in the academic world, addressing the SMP scheduling problem [4]. Several scheduling algorithms have been proposed and most of them came with specific schedulability tests (i.e., means to determine whether a given task set can be successfully scheduled using the given scheduling algorithm). It is difficult to assign a single rate to a scheduling algorithm, since there are multiple traits of it to consider (e.g., how easy it is to implement, how difficult to use its schedulability test is, how much overhead it incurs). However, there is a single factor that better than others measures the worth of an algorithm: its "schedulable utilization".

This term captures the rate at which the scheduling algorithm can use the processingcapacity of the target platform while ensuring that all deadlines can be met; in other words, the highest throughput achievable by the platform (RTOS and processor together) while having a safe and sound system: the higher, the better. The best achievable schedulable utilization (optimality) is when an algorithm can find a valid schedule providing that the total workload is not greater than the total capacity of the platform. Should we really seek optimality? Not really, because in most situations the application load will be considerably lower than the total capacity of the system. However, using an algorithm with high schedulable utilization does have its benefits: the exceeding processing power can be used to enhance the system services or to host additional payload SW, thus increasing the ultimate value of the system; or it is possible to downgrade the platform, thus saving money for the HW and using less energy while the system is deployed.

Analyzing all possible scheduling algorithms proposed in the state of the art is out of the scope of this document. Comprehensive reports exist, which can easily be consulted by those interested. We therefore limit ourselves to briefly discuss the general categories into which the known scheduling algorithms can be subdivided: global, hybrid and partitioned scheduling.

4.2.1 Global scheduling

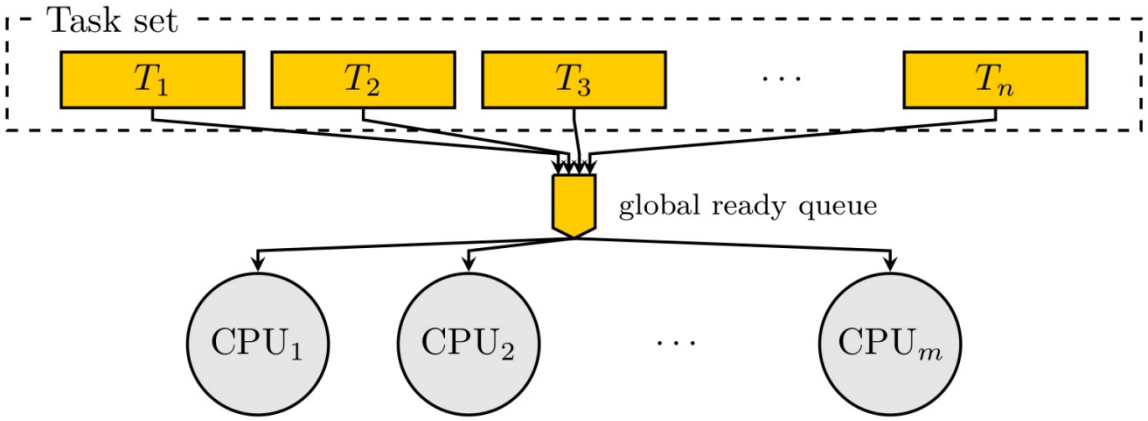

Figure 1: Global algorithm schema.

Global schedulers rely on centralized control: there is one entity that manages all available cores. The specific trait of a global scheduler is that task migration is allowed without restrictions across all cores. As depicted in Figure 1, such a scheduler can be seen as a global ordered queue and a dispatching mechanism that, at given times, takes m elements from the queue and places them into the m available cores. The order of the queue and the dispatching mechanism can be generalization of uniprocessor algorithm (e.g., G-EDF, the global version of Earliest Deadline First, [5]) or based on paradigms specific of SMP platforms (e.g., p-fair [6]). Since the scheduler is in charge of all the cores at the same time, global schedulers are in principle capable of reaching high utilization since, as the runtime workload increases, it can redirect it to any available core. Technically, we say that global algorithms can be work conserving.

However, the possibility for the tasks to execute in any core can potentially lead to an inordinate number of migrations. No known global scheduler has been shown able to avoid (or at least sufficiently mitigate) the negative effects of the migration problem.

4.2.2 Partitioned scheduling

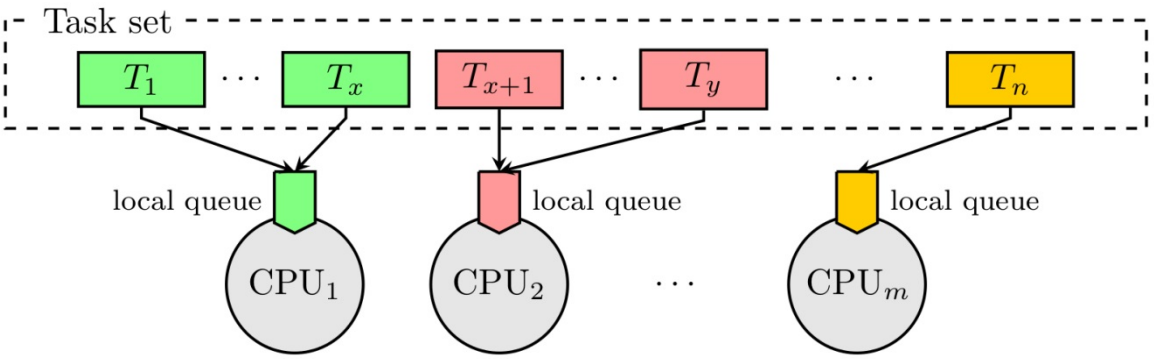

Figure 2: Partitioned algorithm schema.

Partitioned schedulers are similar to the schedulers used for asymmetric multiprocessors: each core has its own scheduler and tasks can not migrate across cores [7]. Whereas in AMPs, cores may even use different instruction sets, on SMPs, there is no physical obstacle that prevents a task from migrating, it is just a logical constraint placed by the scheduler. Since each core represents a stand-alone scheduling problem, it is necessary to decide where each task will execute. The task-to-core allocation problem is analogous to the well-known bin-packing problem, which has no general solution and can only be addressed – in finite time – by best-effort heuristics. In spite of that problem, the use of partitioned scheduling can benefit the tools and the experience coming from the uniprocessor world, so that the balance may still be positive.

4.2.3 Hybrid scheduling

Hybrid schedulers share very little among themselves in terms of common traits. Indeed, all algorithms that are neither global neither partitioned fall in this category. They generally leverage new (i.e., SMP-specific) ideas in which tasks are allowed to migrate, but with some restrictions of variable strength and amplitude.

As an example, there exist algorithms that explicitly select a subset of tasks that are allowed to migrate, while all other tasks are pinned to specific cores (e.g., QPS [8]), and other algorithms define logical servers (as static aggregate of tasks) that become the actual object of scheduling (e.g., RUN [9]), without assigning tasks to cores. The main focus for hybrid algorithms is to find a way to provide a very high schedulable utilization while incurring a low number of migrations.

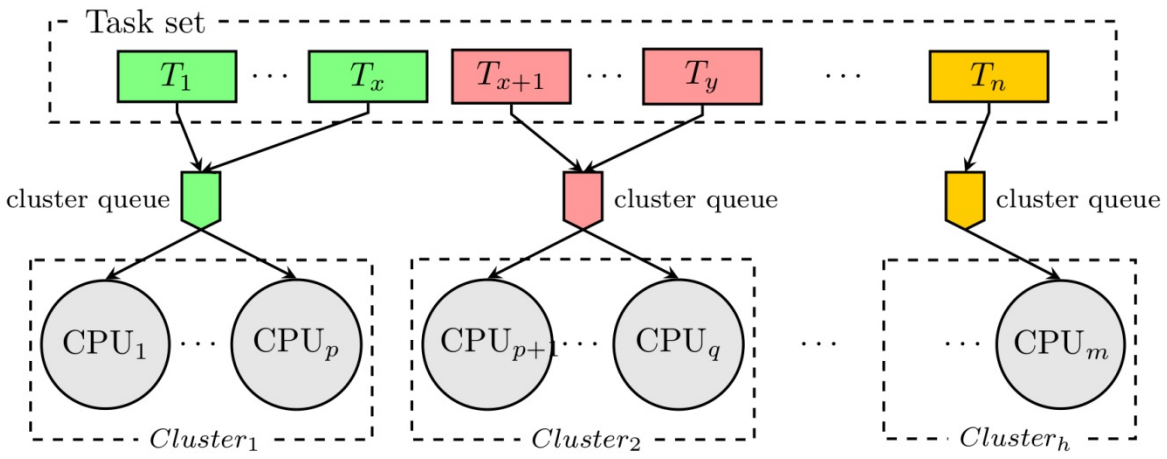

Figure 3: Clustered algorithm schema.

An interesting hybrid algorithm, as seen in Figure 3 is the generalization of both the global and the partitioned approach: the clustered approach [10]. The main idea of the clustered approach is to reduce the scheduling problem into several smaller global scheduling problems: tasks are partitioned and assigned to a cluster of cores, and for each cluster there is an independent cluster-level global algorithm. With this approach the partitioning problem is relaxed, migrations are more controlled (a task can migrate only in a specific subset of cores), and in some platforms the memory topology can be honored (some cores share L2 cache and form a core-cluster, some core-clusters share a L3 cache, etc.).

4.2.4 Our solution

When it comes to considering run-time overhead and time predictability, global scheduling algorithms are clearly not the best solution: inordinate migrations and an efficient way (free from giant locks) to compute a scheduling decisions for the whole platforms are still a problem. Today's global algorithms are still not ready to be used for critical systems. Moreover, global algorithms worsen the already difficult problem of time predictability on multi-core processors. In multi-core processor platforms in fact, parallelism plays against predictability because of the presence of numerous shared HW resources: CPUs are independent, but other HW resources are not (e.g., L2 caches, memory bus, memory controller) and their parallel use causes hard-to-evaluate contention delays. Migrations and frequent global scheduling decisions just increase the stress on the HW (e.g., bus contention for migration, cache disruption when resuming execution).

Hybrid schedulers can offer interesting solutions, but there is not yet enough expertise for their actual use. They lack a solid and exhaustive practical evaluation: most hybrid schedulers, if ever evaluated, are evaluated through simulations. Moreover they lack a complete set of support tools as those developed for uniprocessor schedulers (e.g., schedulability tests, optimization, protocols for access to shared resource, sound ways to account for the scheduler overhead).

In the current scenario, for state of the art and technology baseline, it appears that the best solution is to use partitioned scheduling in conjunctionwith fixed-priority schedulers (P-FP). In addition to preserving all the relevant uniprocessor knowledge, it should also lead to a predictable runtime behaviour that should be similar (with respect to delays and overhead) to the one observed in single-core platforms.

Interestingly, implementing P-FP in the retargeting of RTEMS for an SMP asks for a full-redesign of RTEMS. Indeed, even if RTEMS already uses uniprocessor scheduling, the requirements to implement it for an SMP are quite different: memory is now being shared among parallel cores and there is no single RTOS-and-executable bundle per core; hence, all kernel data structures must be centralized. As a first approach, a giant lock can be used to synchronize all kernel procedures, but in order to have an efficient OS, it must be removed. An interesting observation is that using a partitioned scheduler is similar to using processor affinities with only two constraints: the affinities do never change and each task is affine to exactly one processor. Processor affinities when used in an arbitrary way (APA) are not a valid solution for critical systems: there is still insufficient knowledge that can be used to safely ascertain the stability of the system [11]. In fact, APA is just a theoretical concept stating that tasks have a predefined pool of processors in which they can execute. Before using it for real, major decisions must be made to define the runtime behavior of the scheduler.

- *Weak APA invariant:* running tasks never migrate. This is sub-optimal and hardly improves over the partitioning scheduler: it can slightly increase the throughput of the system, but it buys the overhead due to manage (possibly overlapping) clusters of processors and the difficulty of determining the schedulability of the system.

- *Strong APA invariant*: running tasks can migrate to make room for higher priority tasks that have no other available processor where to execute. This is a powerful property that is, however, very difficult to exploit: its power stems from the fact that affinities can create partially overlapping clusters of processors where different tasks can execute. This leads to maintaining at run time a dynamic-sized set of tasks that can interfere with each processor, possibly reducing the load on overloaded processors. However, considering today's body of knowledge, there are several major downsides with the strong APA invariant: there is no easy way to determine the best possible assignment of affinities to tasks such that the throughput is maximised; APA can in fact induce long chains of migrations, and the overhead to compute the

scheduling decision can be non-negligible. Indeed, also modern general-purpose OSes do not completely adhere to this paradigm: for example, the Linux kernel uses a relaxed version of the strong APA invariant, to decrease the computational cost when performing scheduling decisions and to limit the number of migrations.

4.3 Communications

Independently from the scheduling algorithm used, it is important to determine and offer a sound way to let tasks communicate: while in a multi-core environment there is the actual need to use mechanisms that can handle actual parallelism, and there is more than just concurrency (in contrast with single-core environments). As an immediate extension from the general AMP, events and messages can be successfully used without a complete re-design. However, messages and events in an SMP are best used as a simple synchronization method, where there is no or little data to transfer. Indeed, as memory is shared among cores, it is hardly an option to not offer a way to share it in a safe way, so as to allow several tasks to share a possibly large amount of data without requiring it to travel as a message. Sharing memory normally means using semaphores to regulate access to it. Semaphores in multi-core processors are delicate. For low level synchronization, where the critical sections are expected to be short and error-free, ticket locks can suffice, offering a light-weight and fair synchronization method [12]. However, for application-level critical sections they cannot be used since they could prevent dispatching for long periods.

In the literature, several semaphore-based high-level protocols have been devised for multiprocessors. In fact, only two of them have optimal characteristics: OMIP (O(m) Independence-preserving Protocol) [13]) and MrsP (Multiprocessor resource sharing Protocol) [14]. Optimality for semaphore-based protocols in the scope of multiprocessors is really desirable: having tasks execute on different processors using the same semaphores leads to a considerable increment in the length of the per-processor worst-case critical length, since in the worst case a semaphore must be able to serialize parallel requests coming from different processors. This increment can be suffered not only from tasks using the semaphore, but also from unrelated tasks: optimal protocols are a must so that this problem is avoided or tightly upper bounded.

- MrsP (Figure 4) is built upon the uniprocessor SRP (the generalisation of the priority ceiling protocol) [15], and makes use of spinning in place of suspension, together with a helping mechanism. The helping mechanism assumes that the tasks holding the semaphore can progress the critical section using the spinning cycles of another task waiting for the same semaphore, thus speeding up the release of semaphore. An effective helping mechanism is to let the task holding the resource migrate where a waiting task is currently spinning.

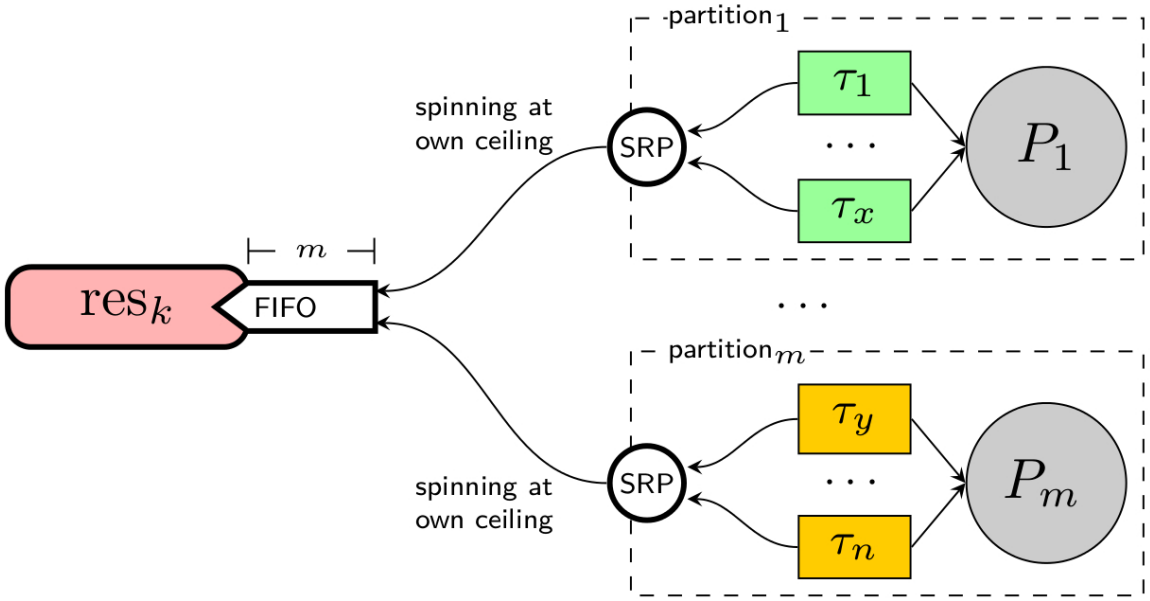

Figure 4: MrsP schema

- OMIP (Figure 5) uses a complex hierarchy of queues, is suspension based and designed to suit clustered systems. In fact, in the partitioned case, this protocol behaves similarly to MrsP. Its helping mechanism requires tasks to migrate tasks.

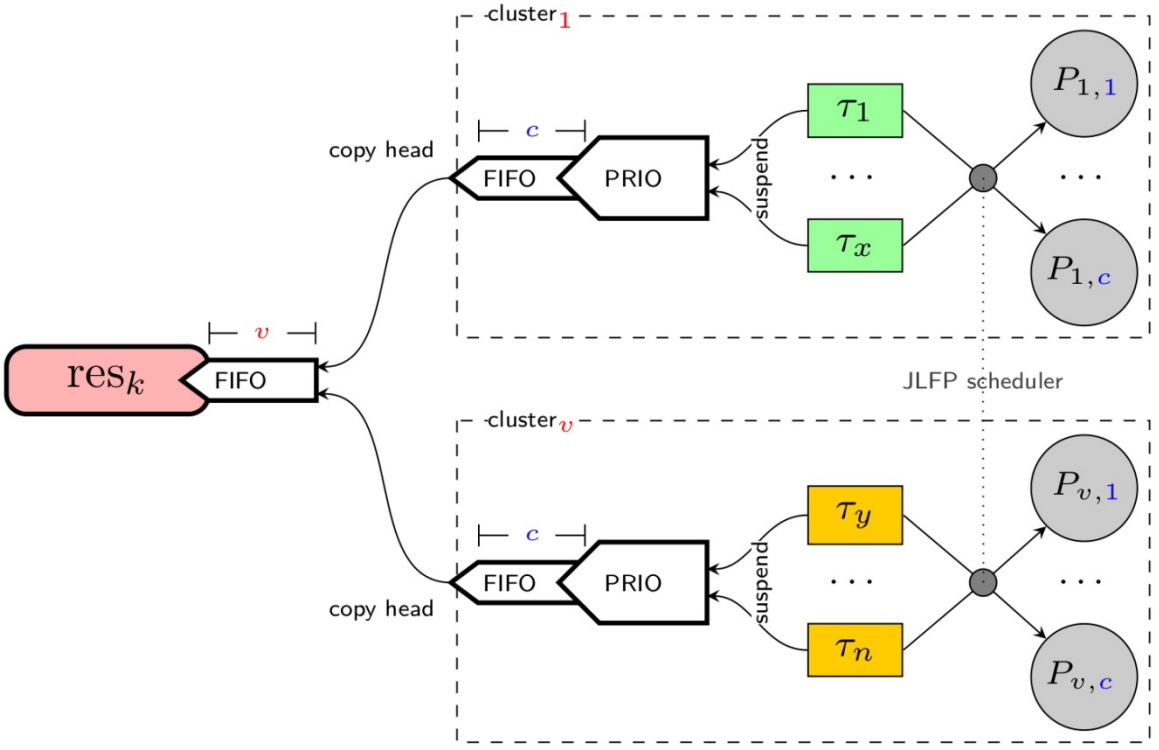

Figure 5: OMIP schema

4.3.1 Our solution

Even if OMIP has promising traits, it is important to note that, from the perspective of critical systems, the advantage of having tasks suspend while waiting for a semaphore, with the consequent runtime improvement on the average responsiveness of the system (assuming that the OMIP hierarchy of queues can be managed efficiently), cannot presently be exploited inside known schedulability tests.

The implementation of MrsP was our first choice in this study, in preference to OMIP, because:

1. it is specifically tailored for partitioned schedulers;

2. it was developed expressly to achieve a graceful extension of uniprocessor's response time analysis, which is a very expressive and sound tool to access schedulability;

3. it is relative simple design should not produce much overhead.

4.4 References

[1] The openMP API, http://openmp.org/wp/

[2] CUDA, nVidia, http://www.nvidia.com/object/cuda_home_new.html

[3] G. Edelin, Embedded Systems at THALES: the Artemis challenges for an industrial group, Invited talk at the ARTIST Summer School in Europe 2009

[4] R.I. Davis, and A. Burns, A Survey of Hard Real-time Scheduling for Multiprocessor Systems, ACM Comput. Surv. 2011

[5] S. K. Baruah, and T. Baker, Schedulability analysis of global edf, Real-time Systems 2008

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, Proportionate Progress: A Notion of Fairness in Resource Allocation, Proceedings of 25th ACM Symposium on Theory of Computing 1993

[7] K. Lakshmanan, R. Rajkumar, and J. P. Lehoczky. Partitioned Fixed-Priority Preemptive Scheduling for Multi-core Processors, 21st Euromicro Conference on Real-Time Systems 2009

[8] E. Massa, G. Lima, P. Regnier, G. Levin, and S. Brandt. Optimal and Adaptive Multiprocessor Real-Time Scheduling: The Quasi-Partitioning Approach, 26th Euromicro Conference on Real-Time Systems 2014

[9] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor, 32nd Real-Time Systems Symposium 2011

[10] B. B. Brandenburg, Scheduling and locking in mulitprocessor real-time operating system, 2011

[11] A. Gujarati, F. Cerqueira, and B. B. Brandenburg. Multiprocessor realtime scheduling with arbitrary processor affinities: from practice to theory, Real-Time Systems 2014

[12] J. M. Mellor-Crummey, and M. L. Scott. Algorithms for Scalable Synchronization on Shared-memory Multiprocessors, ACM Transactions on Computer Systems 1991

[13] B. B. Brandenburg. A fully preemptive multiprocessor semaphore protocol for latencysensitive real-time applications, 25th Euromicro Conference on Real-Time Systems 2013

[14] A. Burns, and A. J. Wellings. A schedulability compatible multiprocessor resource sharing protocol – MrsP, 25th Euromicro Conference on Real-Time Systems 2013

[15] T. P. Baker. A stack-based resource allocation policy for realtime processes Real-Time Systems Symposium 1990

5 PARALLEL LIBRARY STUDY

5.1 Spacebel Approach

Spacebel's basic needs for the parallel libraries were to investigate to exploit the processing power of the various processing cores while maintaining a correct behaviour of the DHS. Therefore, the image processing part of the SWAP instrument has been isolated from the Proba software as to allow for several parallelisation strategies.

5.1.1 Analysis of Existing Libraries

Substantial energy has been spent on analysing parallel libraries such as OpenMP and Cilk Plus.

The first conclusions concerned a series of constraints about the existing libraries:

- They are optimised for target processor architectures, using complex multi-stack structures, while having no tested run-times for relatively unpopular Sparc architectures.
- Try to avoid and bypass OS as much as possible, thereby making them less portable.
- Some libraries have compiler support. We did however not manage to build a gcc compiler that supports OpenMP, RTEMS and the C11 atomic extensions. It shows that gcc support for Sparc might become a problem in the coming years.
- Tend to use active polling as the main goal is to accelerate one single application by exploiting the hardware as much as possible without any real consideration for potential other system work. The latter is no problem in a Linux environment where application tasks have limited priority, in a flight system, the need is to exploit de hardware to improve what could be considered background work in respect to the main DHS job.
- The complexity caused by the various optimisations lead to systems that are not only problematic to port but equally difficult to validate.
- Overall, the investigated libraries show clearly a lack of maturity and proper validation tools that would allow using them in embedded systems

Moreover, as we concluded that porting and integrating an existing library in the demonstrator was not possible at all in the programmatic of the project, we decided to go to the basics, meaning prototyping structures as proposed by the ongoing JTC1/SC22/WG14 – C CPLEX (C parallel language extensions) study group.

5.1.2 Implementations by Spacebel

It should be noted that the primary goal of Spacebel was to investigate the overall Proba DHS gain and behaviour by delegating some of the SWAP image processing to various cores. It would have been nice indeed to have the possibility to extend artificially the payload processing load as to stress the system to a maximum. This would however have required a more detailed demonstrator, test procedures and analysis tools.

The following test cases have been analysed:

- A series of matrix conversions that are split in 4 quadrants
- Proba SWAP image processing where each image has been split in several tiles to allow for parallel processing while maintaining the overall processing work roughly constant.

The limited set of matrix test cases resulted in some surprising results that are summarised in the table below. It separates the average communication cost and the matrix calculation cost per quadrant. For recall, the tests have been run without the caches enabled as the proper cache initialisation was not available at test time and is not automatically done by RTEMS.

Costs in µs	*GR712*	*NGMP 1 worker*	*NGMP 3 workers*
Comms	106	149	284
Matrix	560	74	537

Table 2: Comparison of parallel matrix conversions

The conclusion here is that the lack of cache limits enormously the potential system gain. Both processors have a similar communication costs between cores, the 200MHz NGMP single core performs 8 to 9 times better on the matrix, but loses all its advantages when 4 cores are participating. We suspect that, besides the lack of cache, the various synchronisations and locking was seriously delayed because of the single Giant Lock in RTEMS.

On the parallel libraries for SWAP data processing, we made two different implementations:

- The first one with shared data structures for work distribution and the workers were waiting on a work semaphore, getting job information, processing and signalling the results to the requester.
- The second one was a “discrete” implementation where the main application sends a message to each worker containing the requested processing job and waits till all the requested jobs have been completed.

Both solutions above provided quite comparable Proba demonstrator performance results and analysis of the overall software did not really provided the needed to analyse the details of the various components. Therefore, the final Proba demonstrator campaign has been limited to the the discrete implementation.

On the Proba SMP demonstrator, results and conclusions are quite different, mainly due to the more coarse grain granularity of the image processing. The distributing on the several NGMP cores generates an 2,4 % CPU overhead, while on the GR712, it adds 28 % overhead. The differences are attributed to the far more efficient 128-bit bus that loads in one access a complete cache line, which is quite important for applications that process larger data sets.

The first observation is that communication cost is much higher than expected and scales not well at all. This is probably much aggravated by the fact that there are several protected regions in the data structures, which makes it even worse. It is an example where the Giant Lock used in the current version of RTEMS SMP can quickly degrade the system as more cores get involved.

It is very clear that the matrix computation creates a very high interference on the shared memory bus. To be able to perform a better analysis, code should be more instrumented to maintain more

detailed statistics and worst case behaviour. One should normally expect that worker 1 should process the fourth frame. It looks as if they are done by the second worker, so some system asymmetry makes that worker 2 performs the fourth frame: not sure if this is caused by priorities on the memory bus, unfair queuing on the message queue or semaphores.

There is however a lot to be learned from the implementation.

A mono-processor design that is correct on paper, doesn't scales always well on a multi-core system. The more there are protected zones and synchronisation points, the more the performance degrades as "system" activity increases.

RTEMS SMP in its current form is usable provided the Giant Lock, and hence the services that rely on it, are avoided. This situation should be substantially improved when RTEMS SMP will use a finer grain locking mechanism and when a more efficient inter-processor queuing mechanism is implemented that allows for a better decoupling of the cores. Typically, a mechanism that allows for simultaneous writing and reading on a queue would already help a lot.

When looking in the code of other parallel libraries, such as in Cilk, Go, OpenMP, the (nested) blocks used in Grand Central Dispatch from Apple, it becomes quickly clear that none of them simply inherit the approaches as traditionally used in RTOS environments; they are the result of careful thinking and incremental development and tuning as to exploit fully the hardware capabilities. The best way to demonstrate this is to look into http://lmax-exchange.github.io/disruptor/files/Disruptor-1.0.pdf that shows that basically all habits have to be questioned and revisited. On the other hand, there exist many dozens of parallel languages and application libraries, which demonstrates clearly that there is no single best solution.

5.1.3 Conclusions

So basically, with some RTEMS improvements (and extensions) and tuning, it is believed that the parallel library overhead could be reduced to the 50 µs range while using up to 4 cores. It might be possible to reduce that further with another 50 % when completely rethinking and developing from the ground up.

Before spreading out an application over several cores, a number of questions need be asked:

- The costs of parallelisation amounts easily to 50 µs, so the parallel tasks should take more than 100 µs before it is worth it,
- Parallel processing efficiency is very much related to the used bus bandwidth and cache efficiency, where the NGMP excels.

Anyway, further evaluation and analysis makes only sense if the basic RTEMS mechanisms are updated to exploit its partitioning (and uncontended locks) along with fine grain locking.

Finally, another aspect needs to be considered for space platform controllers; by delegating non time critical processing work to other cores, we can gain substantial processing head-room and potentially determinism on the main processor.

6 PROBA DEMONSTRATOR

6.1 Approach

The test approach consists in executing a timeline in which series of TCs start or activate the functionalities of the demonstrator at a specified time. In term of functionalities, they are mainly linked to the PROBA-2 applications (image acquisition, processing and compression). In this sense, each timeline is a scenario in which the expected results (explicitly described in the plan) are compared with the values obtained via TM.

The test cases are executed automatically but the major part of the verification is done manually.

Several types of requirement are verified in the tests; performance, interface, initialisation or functional requirements are verified.

Along the test campaign, the demonstration software remains the same for each scenario but it differs depending on the execution target.

The complete software architecture has been overhauled to make it more modular and most importantly, to allow to assign the processor core to each of the software components. This means that we can position any Proba DHS component on any core of the processor just by configuration.

6.1.1 SWAP reference image

The demonstrator implements the main manager of PROBA-2 SWAP payload (SWAP manager is above the DHS layer). The SWAP manager handles the acquisition of SWAP images and it manages the instrument mode and a fortiori the instrument itself. On demand, the acquired images can be processed and compressed in JPEG. Therefore, JPEG algorithm must be fed with data in accordance first SWAP specification and the specification of JPEG algorithms (which is a bit particular in PROBA-2).

For this purpose, a reference image is used to feed SWAP manager and the JPEG algorithm. The reference image comes from the formal validation tests of PROBA-2 OBSW. For the record, at the time of PROBA-2 qualification and development, no SWAP image material exists. Therefore, a couple of reference images have been chosen for the tests.

6.1.2 Test Scenario Approach

All the test cases, the sources, the configurations are stored under SVN configuration. A clean checkout from SVN is performed and then, before the test campaign itself, the image is rebuilt via Makefile depending on the test platform.

The execution of the test cases is automated via scripts. The timeline and the image are loaded in the RAM memory of the target. When the loading is completed, the software is started at the entry point specified by the test script. All formal tests have been run on GR712 and NGMP 200 MHz on mono and multi-core configurations as to be able to compare relative performance and synchronisation cost.

At the end of the execution, the TMs are dumped from the target and the TM are decoded automatically and the results depicted manually against the plan and the expected results. At the same time, the resulting CPU load per task is dumped in a log file.

The resulting image slices are manually recombined and verified against the original processing results.

While various combinations of DHS components on various cores have been tested, only the configurations that run the main components on the first core and the image processing worker threads on the other cores resulted in a consistent success rate on all configurations. Runs with cache disabled failed as well as the resulting performance was no longer sufficient to satisfy the test requirements.

All formal test using the two latest RTEMS toolkit releases finally run OK.

6.2 Demonstrator CPU Reports

While the previous tests contain a quite exhaustive functional testing, we are interested to have more information about load balancing and the cost of distribution. Therefore, most tests have been added informally to be able to compare with mono core versions.

6.2.1 Basic CPU Usage Analysis

For recall: GR712 at 48 MHz and using SDRAM, NGMP at 200 MHz.

Comparisons have been made with identical RTEMS SMP, compilers and demonstration software. RTEMS SMP is however slightly different from GR712 and NGMP due to other memory mapping and BSP. For the demonstrator, it was rather problematic to do comparisons between the classic RTEMS and RTEMS SMP in a mono-core configuration.

The differences between mono and multi-core execution are only caused by placing (through affinity) the worker components on other cores and by adding a BGDn task on each core that has a worker thread to replace basically the idle task and to be able to measure the idle time.

The BGDn tasks contain a loop forever that put the CPU in power down mode. This allows to verify that the power down mode works correctly, but more importantly that the processor is woken-up as needed.

Enabling caching on the NGMP is not done by default in the RTEMS SMP configuration. The demonstrator failed when doing tests without the cache being enabled on the NGMP because of missed deadlines, so results have not be formally reported.

Enabling NGMP cache increases image processing speed with a factor of 12,5. On the DHS, speed improvement is only an average factor of 5, probably because the DHS uses many RTEMS calls that take less profit from the caches.

This leads however to a first interesting global conclusion: isolating long processing tasks in non-interrupted cores results in better exploiting the available processing power.

6.2.1.1 Single Core versus Multi-core

Boot time increases as the first core needs to be properly initialised before the other cores are enabled. For the demonstrator, time since the last CPU usage reset increased with 20 milliseconds on the GR712, 114 milliseconds on the NGMP.

De following lists details the additional time needed when switching from a mono-core to a multi-core processor:

Task	GR712	NGMP
EVNT	-4 %	-10,2 %
WRKn	4,6 %	8,85 %
SCHD	-4,44 %	-4,73 %
TCHD	4,14 %	1,88 %
TMTC	4,07 %	1,15 %
HK	5,23 %	-1,36 %
OBSR	2,27%	1,35%
OBSS	1,12%	-0,53%
MONI	5,02%	15,17%
DAM	2,04%	0,30%
SWDM	-0,39%	8,54%
SWIM	27,75%	2,35%
MIMC	1,92 %	0,01%
SYSM	4?46 %	18,43 %

Separating DHS, that contains many small tasks that are using relatively intensively RTEMS, and the computing intensive worker tasks show almost an identical aggregate CPU throughput. On the GR712, the workers require 5 % more time on the second core, which is good considering the fact that all data has to be reloaded in cache and that some of the parallel processing data structures are protected by semaphores. On the NGMP, the total time of the workers on the three cores increases only with 8 %.

Basically, on average, the DHS part of the demonstrator loses 3 to 4 % performance, which we think we can attribute to the fact that all RTEMS activities have to go through the giant lock, combined with potential semaphore contentions in the worker threads and cache snooping artefacts. While the average performance drop is similar on both systems, there is however a significant different performance drop for specific tasks.

Monitoring and Sysmanager drop 15 to 20 % in performance on a multi-core NGMP processor, and only 4 to 5 % when switching from mono GR712 to multi-GR712. It is most probably related to the fact that the Sysmanager and Monitoring do a tremendous amount of semaphore locking, so it can most probably be attributed to the fact that NGMP loses some of its performance gain when doing system work that involves several cores.

To the contrary, SWIM has only a 2,4 % performance penalty on the NGMP when switching to multi-core, while it drops 28 % on the GR712. This seems related to the fact that SWIM reads all image data, while the workers on the other core are invalidating and caching the same data. Here we can conclude that the NGMP 128 bit data bus helps the cache management significantly, and hence providing a better balanced system.

6.2.1.2 GR712 versus NGMP

When isolating the DHS related figures from the image data processing figures, we come to the conclusion that:

The NGMP at 200 MHz is naturally much faster that the GR712:

- 4,25 times faster on DHS related activity,
- 4,41 times faster for payload data processing.

So basically, the NGMP scales very well with its clock speed increase. The fact that the cores have to share the same memory is very well compensated for by the 128 bit wide bus that fills up a cache line in one single access cycle.

6.2.2 Conclusions on RTEMS SMP on Multi-core Processors

The demonstrator approach allowed for a quick feeling about the multi-core potential. The results show that the additional processing capability, which is more important than we expected, probably outweighs the uncertainty caused by the much more complex interactions and interferences caused by the shared memory bus. The project confirmed our hope that multi-core processors running RTEMS SMP have the potential to use such a system where each core is dedicated to a certain job without being impacted too much by the other core.

The demonstrator has been structured in such a way that it is in principle, easy to reallocate tasks to another core without changing the software. In practise however, there are many protected regions used in the DHS design; this tight coupling makes that moving such tightly coupled task to another core makes that the gained performance will be outweighed by the additional RTEMS communication overhead and subsequent non determinism and system complexity.

But before investigating further, especially in the light of determinism and worst case analysis, it might be better to improve first a number of essential aspects.

- Refine the RTEMS “Giant lock”, as in Linux, into deeper and more specialised locks that suffer from much less contention. This should allow avoiding involving other cores unless strictly needed.
- Design the software in such a way that they are loosely coupled between cores, for example by message passing. The current demonstrator inherited several protected areas using semaphores; very efficient in mono core systems, a source of potentially unbounded response times in multi-core systems.

Further investigation around the RTEMS message queuing functionality, using parallel library matrix conversions (but unfortunately with cache disabled), showed that on average, the GR712 bi-core needs around 100 µs to send a message to a queue and resynchronise, the NGMP in a bi-core configuration needs 149 µs. Problem however is when 3 cores are waiting on the same queue:

average communication costs raises to 250 to 330 µs. So that confirms a number of our anticipations:

- One to many communications are having a raising overhead as more cores are involved and become less predictable.
- The RTEMS message queuing functionality scales not very well for multi-core usage and might need a complete revision.

For the latter point, we think that having an additional RTEMS simple queueing service might be a good idea:

- No need for priority, time-out, very long messages, no suspension when writing on a full queue
- Might be limited to point to point,
- Could be designed to allow simultaneous writing and receiving, so better decoupling and much shorter locking (the current version is most probably locked for the whole operation, a kick-off of the receiving end might be the only operation requiring a lock)
- So overall, this type of operation could most probably be reduced to a predictable couple of tens of µseconds.

In the long run, it should be possible to have critical RTEMS SMP based systems that run almost without contention between cores. Further test cases are needed to investigate then the worst case scenarios. It might be a good idea to investigate further core decoupling for several reasons as discussed in the following section.

Overall thread dispatch disable durations in the order of hundreds µs have been observed, although this should decrease rapidly as fine grain locking and partition isolation improves. At first glance, interrupt disable times seems better, but this needs further instrumentation and testing, by preference with real IO activity as drivers are very often the source of interrupt locking.

6.3 Programmatic Conclusions

We feel that the meaning and advantages of partitions is understood differently by different actors.

For Spacebel, a partition is something that is functional wise is completely decoupled from another partition; something that happens in one partition should never impact significantly the behaviour from another partition (except the shared memory timing interference). To improve such partition isolation, RTEMS should not only move towards finer grain locking, but each partition scheduler should have its own locks, its own timer interrupt, probably a better and independent watchdog driver, ...

While we lose some level of determinism at the microscopic level, on the macroscopic level, we can gain significant overall system complexity reduction, thereby gaining in system determinism, validation, maintenance and reuse. The potential core “shielding” by dedicating tasks and interrupts to a specific core opens the door for a better partitioning approach with the best compromises concerning the various constraints. So by loosely coupled cores and RTEMS SMP, we increase significantly the system capability without the exponential complexity explosion.

Basically, a DHS OBSW is a complex piece of code with hundreds of thousands lines of code that are difficult to maintain and validate. Proper time segregation of such a partitioned system allows adding ancillary system components that can be validated in a separate way. It is true that in that case, space isolation is only be design and not enforced, but allows nevertheless to approach the advantages of an TSP/IMA system without its constraints. This is especially true if we want to further build on the various field proven DHS systems.

The next logical steps would be to have MMU support in RTEMS for improved space partitioning. Alternatively, it might be a good idea to isolate a number of cores using a Time and Space Partitioning hypervisor. This would provide an evolutionary growth path for better partitioned systems without having to rework completely the “trusted and proven” data handling systems that are flying today.

7 PROJECT CONCLUSIONS

Spacebel has been impressed by the work and dynamics of all the team members, including Estec and Aeroflex Gaisler (AG). AG was very impressive doing all final integration and test work.

The 200MHz NGMP multiplies the available performance for the DHS with a factor of 4 to 5, while the other cores give some spare processing power of 2 to 3 additional cores without necessarily impacting dangerously the dynamic behaviour of the first core.

If RTEMS SMP partition isolation can be improved along with loosely inter-partitioning coupling, we would not hesitate to recommend the GR740 platform running RTEMS SMP.

By restraining the scope of RTEMS SMP to one single partition (and core), a first validation/qualification campaign could be started to get already a qualified baseline. This is typically a case where a qualified baseline, along with its procedures should be published in the public domain, so that RTEMS evolutions could be introduced and qualified in an incremental way. Avoiding orphan qualified versions might seem more economic in the short term, it is certainly a better solution in the long run.

It needs still some analysis and study work to investigate how RTEMS SMP can be qualified on multi-core system without exponentially increasing the efforts.

Last page of:
35 pages

Program:
TRP

Project:
Development Environment for Future LEON Multi-Core

Document
RTEMS SMP Final Report

Reference:
SPB-DE4FLMC-301-FR-001

Issue:
1 - 20/04/2015

Revision:
0 - 20/04/2015

Distribution Code:
Restricted Distribution

Spacebels,a,/n,v,
BE-0435,536,532

Web : www,spacebel,be
Mail : info@spacebel,be

Liège

Liège Science Park,
Rue des Chasseurs Ardennais, 4
B-4031 Angleur (Belgium)
Tel : + 32 4 361 81 11
Fax : + 32 4 361 81 20

Brussels

I, Vandammestraat 7
B-1560 Hoeilaart (Belgium)
Tel : + 32 2 658 20 11
Fax : + 32 2 658 20 90

Toulouse

Spacebel France
6, Voie l'Occitane
BP 87671
F-31676 LABEGE CEDEX (France)
Tel : + 33 5 61 00 35 44
Fax : + 33 5 61 00 20 31