Multiprocessor Ada platform based on MaRTE OS and GNAT[1]

Mario Aldea Rivas, Héctor Pérez Tijero and Michael González Harbour
Universidad de Cantabria
39005-Santander, SPAIN
{aldeam, perezh, mgh}@unican.es

Abstract:

This paper presents the implementation of a multiprocessor Ada platform based on MaRTE OS and GNAT. The paper describes the modifications required to adapt MaRTE OS to a multiprocessor architecture, namely the protection of the internal data structures against concurrent access and the management of several ready queues. We also describe the adaptation of the GNAT run-time library in order to be used on top of the multiprocessor services provided by MaRTE OS. The developed MaRTE/GNAT platform provides full Ada tasking functionality and the multiprocessor support defined in the Ravenscar profile. Two multiprocessor architectures are supported: the multiprocessor version of the XtratuM hypervisor for Intel x86 architecture and the Linux operating system (with MaRTE taking the role of a Pthreads multiprocessor library).

1. Introduction

Multiprocessor architectures are becoming popular in many application areas including real-time and embedded systems. A particular kind of multiprocessor architecture is the "Symmetric multiprocessor" (SMP), where there is a single memory space shared by two or more identical processors.

The Ada language, in its 2012 revision [1], has included support for SMP systems. The core of the Ada SMP support is the ability to allocate tasks to a set of processors grouped in a "Dispatching Domain" or to a specific processor. Ada also allows migration of tasks between processors and dispatching domains. This flexibility allows supporting a large range of allocation approaches: from fully partitioned systems where each task is allocated to a single processor on which all its jobs must run, to global allocation systems where all tasks can run on all processors and jobs may migrate during execution.

Ada allocation of tasks to processors is based on the *Dispatching Domain* concept. Dispatching Domains are disjoint sets of processors defined by the application at elaboration time. Every task is allocated to a dispatching domain. Inside its domain, a task can execute in any processor unless it is explicitly assigned to a particular processor. The processor affinity of a task inside its domain can be changed at run-time as many times as desired.

In Ada 2012, the Ravenscar profile has been extended to support multiprocessor systems using the fully partitioned approach ([1] D.13.1(8-9/3)). In this restricted allocation model, there is only one dispatching domain that includes all the processors in the system but, according to the profile rules, the application must assign every task (using the CPU aspect) to the processor where all its jobs will be executed.

The GNAT compilers (since the 2012 version) provide full Ada multiprocessor support, i.e., arbitrary number of dispatching domains, global and per-processor task allocation and task migration. This support is also included in the free GNAT Academic Program (GAP) compilers.

XtratuM is a hypervisor specially designed for real-time embedded systems developed at the Real-Time Systems Group of the Instituto de Automática e Informática Industrial of the Universitat Politècnica de València (Spain) [2]. The hardware platforms supported by XtratuM include the Intel x86 SMP architectures. When running on that architecture, XtratuM allows the execution of isolated multiprocessor applications to be executed running on top of different operating systems (e.g., Linux or MaRTE OS).

MaRTE OS [3] is a real-time operating system developed by the Software Engineering and Real-time group at the University of Cantabria. Most of its code is written in Ada with some C and assembler parts. The Gnat Run-Time Library (GNARL) has been adapted to run on top of MaRTE OS in order to provide full Ada tasking support. The MaRTE/GNAT platform implements most of the advanced Ada real-time services defined in the 2005 and 2012 revisions of the standard [4]

1. This work has been funded in part by the Spanish Government and FEDER funds under grant number TIN2011-28567-C03-02 (HI-PARTES)

[5]. Previously to the realization of the work presented in this paper, MaRTE was a uniprocessor OS for Intel x86 architectures and there was a port of MaRTE to the uniprocessor version of XtratuM which was used in [6].

In this paper we present the adaptation of MaRTE OS to multiprocessors. Two multiprocessor architectures are supported: the multiprocessor version of the XtratuM hypervisor for Intel x86 architecture and the Linux operating system with MaRTE taking the role of a Pthreads (POSIX-threads) multiprocessor library. Our MaRTE/GNAT Ada platform supports the restricted multiprocessor support included in the Ravenscar profile, with static allocation of tasks to processors.

The document is organized as follows. In Section 2, we present an overview of the relevant aspects of the SMP architecture from the operating system point of view. Section 3 describes the implementation of the multiprocessor support in the MaRTE/GNAT platform. Details about the porting of MaRTE OS to XtratuM are provided in Section 4. Section 5 describes the adaptation of the multiprocessor version of MaRTE in order to act as a POSIX-thread library for Linux. Finally, Section 6 gives our conclusions and outlines some research lines that arise from this work.

2. Overview of the SMP architecture from the operating system point of view

In an SMP architecture, the existence of a single memory space makes it possible to execute a unique copy of the operating system. The system calls are typically executed by the processor where the call is performed, which implies that the operating system data structures must be protected against concurrent accesses from threads running in different processors, as it will be discussed in Section 3.1.

Besides the single memory space, there are two other relevant aspects to consider from the point of view of an SMP operating system: the available inter-processor communication mechanism and the way the different processors are initialized.

The typical inter-processor communication mechanism is based on inter-processor interrupts (IPI). The IPIs are a special type of interrupt that allow a processor to interrupt and force the execution of a handler in another processor. A multiprocessor operating system uses IPIs to force a dispatching operation in a different processor, for example after changing the priority of a thread in a different processor than the one that is executing the priority change operation.

In most SMP architectures, and in particular in the Intel x86 SMP, initialization is performed in two stages. In the first stage, just following a power-up or reset, a particular processor is selected by the system as the bootstrap processor (BSP). The BSP is in charge of executing the initial configuration of the OS. Meanwhile, the other processors, designated as application processors (AP), remain in the halt state. During configuration, the BSP installs the interrupt handlers, including the dispatcher interrupt handler. These handlers will be shared by all the processors in the system. Once the initial configuration has finished, the BSP starts the APs which execute a simple piece of code that enables the local interrupts, sets a global variable to notify they are ready and waits in an infinite loop.

In the second stage, that begins once all the APs are ready, the BSP sends to each AP the initial dispatching IPI. The first time the dispatching handler is executed in an AP it jumps to the idle task context. When all the APs have been correctly initialized and are executing their respective idle tasks the initialization phase has finished. From that point on, subsequent IPIs will be used to dispatch the application threads in the different processors.

3. Implementation of the Ada multiprocessor support

3.1. Support at the operating system level

In this section we present the changes performed in MaRTE OS (originally designed as a uniprocesor operating system) in order to adapt it to a multiprocessor architecture. Instead of performing a full redesign of MaRTE from scratch our goal has been to reuse most of the MaRTE internal structure and identify the minimal changes required to adapt it to a multiprocessor architecture.

Two are the main issues to be solved:

a) The management of several ready queues.

b) The protection of the operating system data structures.

The requirement of managing more than one ready queue arises from the allocation model we want to support in which every task is assigned to one, and only one, processor. An efficient implementation of this allocation model requires the use of one ready queue and one idle task per processor. The data structures for the ready queue and the idle task control block

along with the reference to the local running task are stored in the "Processor Control Block" (PCB), a software entity associated to every processor in the system.

In order to protect the operating system data structures against concurrent accesses from threads running in different processors a set of spinlocks has been created. We have taken an intermediate solution among the two extreme locking strategies: the coarse-grained locking, where a unique lock is used to protect all the operating system data structures, and the fine-grained locking, where a different lock is used for each data structure. The coarse-grained locking is easier to implement but reduces the concurrency, while the fine-grained locking improves the concurrency at the expense of increasing the complexity and the risk of deadlock.

In our solution all the systems calls are protected by a unique kernel lock. The use of this coarse-grain lock implies that while a task (or an interrupt handler) is executing a system call in one processor, any other system call in another processor will be blocked and the entity (the task or interrupt handler) invoking the call will spin waiting for the first call to finish. In order to reduce the duration of this global blocking a different lock is associated with each PCB. Taking advantage of these PCB locks, long system calls affecting two processors are divided into two parts: the first one, performed with the kernel and the PCB locks taken, is executed in the processor where the call has started; and the second part is executed in the processor where the system call has effect but only with the corresponding PCB lock taken, and with the kernel lock released.

The low-level abstract interface for accessing the hardware (HAL, Hardware Abstraction Layer) of MaRTE OS has been extended to include operations related to the management of multiprocessors. The more relevant operations are grouped in the package MaRTE.HAL.SMP_Interface shown in Figure 1.

```
package MaRTE.HAL.SMP_Interface is

   type CPU_Range is range 0 .. Integer'Last;
   for CPU_Range'Size use 32;

   subtype CPU is CPU_Range range 1 .. CP.Num_Of_Proc;
   type CPU_Array is array (CPU'Range) of Boolean;

   function Number_Of_CPUs return CPU;                          -- number of processors available to the application

   function Get_CPU_Id return CPU;                              --  returns the identity of the CPU on which the call is being executed

   procedure Send_IPI (CPU_Id : in CPU; IPI : in Integer);      --  to send an inter-processor interrupt

   procedure Set_CPU_Ready (The_CPU : CPU);                     --  to notify that a processor has completed the initialization process

   function Is_CPU_Ready (The_CPU : CPU) return Boolean;        --  to know if a particular CPU has completed the initialization process

   procedure Set_CPU_Ready_To_Receive_First_IPI (The_CPU : CPU);--  to denote that a CPU has been started up

   procedure Initialize;                                        --  to initialize the data structures and start up the available processors

end Marte.HAL.SMP_Interface;
```

Figure 1. SMP management operations for the adaptation of MaRTE OS

The particular implementation of the SMP operations relies on the primitives provided by the underlaying platform (i.e. XtratuM or Linux) as will be presented in Section 4 and Section 5 respectively.

3.2. Multiprocessor application programming interface

The POSIX standard does not provide any interfaces to assign threads to processors, and consequently a POSIX operating system must define its own extensions. For this purpose Linux defines the functions pthread_attr_setaffinity_np() (to specify the set of processors assigned to a thread at creation time) and pthread_setaffinity_np() (to dynamically change the affinity of an existing thread). Linux supports a complex allocation model where threads can be assigned to any arbitrary set of processors and the assigned set can be changed dynamically during the life of the thread.

The Ravenscar profile allows a much simpler allocation model where every task is statically assigned at creation time to one, and only one, processor. The extension of the MaRTE API follows this model and consists only on adding a new thread attribute called "processor affinity" to the thread attributes object. This attribute is managed with the MaRTE specific functions:

```
#include <pthread.h>
int mthread_attr_setprocessoraffinity(pthread_attr_t *attr, int proc_id);
int mthread_attr_getprocessoraffinity(const pthread_attr_t *attr, int *proc_id);
```

The default value for the "processor affinity" attribute is the Bootstrap Processor.

Other functions required to provide support for the Ada multiprocessor packages are mthread_getprocessoraffinity(), to retrieve the processor affinity of an existing thread, and msched_getnprocs(), to get the number of available processors:

```
#include <pthread.h>
int mthread_getprocessoraffinity(pthread_t thread, int *proc_id);

#include <sched.h>
int msched_getnprocs();
```

3.3. Modifications to the GNAT run-time

The GNAT run-time system adapted for MaRTE OS is based on the Linux version provided by AdaCore with the GAP compilers. The GAP compilers provide full Ada multiprocessor support since the 2012 version, i.e., they support an arbitrary number of dispatching domains and global and per-processor task allocation. In our work, this flexibility is restricted to the fully partitioned allocation model where there is only one global dispatching domain, every task is allocated to one and only one processor, and no task migration is allowed. According to these limitations, the CPU aspect is the only multiprocessor Ada aspect supported, and applications cannot use the Dispatching_Domain aspect nor depend on the System.Multiprocessors.Dispatching_Domains package.

The original multiprocessor support provided by the GAP compilers is based on the Linux services to assign threads to sets of processors through the function pthread_attr_setaffinity_np(). The implementation of the CPU aspect in the run-time creates a processor set containing just the assigned processor and uses it as a parameter for the pthread_attr_setaffinity_np() function.

In our modification of the run-time, the set creation is avoided and the assigned processor is directly used as parameter for the MaRTE OS specific function mthread_attr_setprocessoraffinity() presented in the previous section.

4. Porting MaRTE to XtratuM

The XtratuM hypervisor uses para-virtualization [2] techniques to provide partitions with isolated execution environments. Under the para-virtualized model, the guest operating system is aware that it is running within a virtualized environment and therefore it has been modified to access the hardware by means of the para-virtualized operations or hypercalls.

In the case of XtratuM, these hypercalls are as close to the hardware as possible so that porting an operating system that has already worked on the native system only requires replacing some parts of the operating system with the corresponding hypercalls. Since MaRTE OS defines a low-level abstract interface for accessing the hardware (HAL, Hardware Abstraction Layer), the required modifications are strictly located in this interface as described below.

4.1. Interrupt handling services

As XtratuM is responsible for handling all the available hardware interrupts, the hypervisor provides guest operating systems with a virtual interrupt table. Once a hardware interrupt has been triggered in the system, the hypervisor will subsequently generate the appropriate virtual interrupt to the target partition. As a result, MaRTE OS cannot directly enable or disable hardware interrupts, but it should enable or disable the corresponding virtual interrupt using the hypercalls provided by XtratuM.

Furthermore, the hypervisor also provides partitions with a special kind of virtual interrupts called extended virtual interrupts, which allow the guest operating system to be notified about specific events related to the XtratuM temporal and spatial partitioning.

4.2. CPU management services

Similarly to any virtualized resources, XtratuM virtualizes the available CPUs and provides partitions with virtual CPUs and specific hypercalls for their management. Among others, the hypervisor defines a hypercall to send inter-processor virtual interrupts (IPVI). Support for the management of multiprocessors in MaRTE OS has been built upon this mechanism as described below:

- **Notification of scheduling changes.** The IPVI mechanism enables guest operating systems to inform the virtual CPUs about dispatching events. In the case of XtratuM, IPVIs are sent to a target partition and therefore all the virtual CPUs

associated with this partition will receive the IPVI. However, dispatching operations are intended to be executed on a specific virtual CPU, and therefore MaRTE OS must keep track of the target virtual CPU and dismiss spurious interrupts in the remaining virtual CPUs.

- **Initialization process.** XtratuM chooses one of the virtual CPUs as the BSP and leaves the responsibility of starting up the remaining virtual CPUs (APs) to the guest operating system. For the adaptation of MaRTE OS, the initialization is a two-stage process. In the first stage, the BSP starts the APs using a specific XtratuM hypercall (`XM_reset_vcpu()`). The second stage begins once all the APs are ready. Then, the BSP sends an IPVI to perform a context switch and start the execution of the idle thread in each AP. From this point on, the initialization process is considered completed.

4.3. Time management services

Since clocks and timers represent shared resources for partitions, guest operating systems must access these services through the corresponding hypercalls. XtratuM provides partitions with two kinds of software clocks: the first clock is associated with the native hardware clock, which is global for all the virtual CPUs; the second clock is associated with the execution time of each virtual CPU. In this work, for implementation simplicity, only the former clock is supported by the adaptation of MaRTE OS to XtratuM.

5. MaRTE OS MP as a Pthread library for Linux

MaRTE OS can be configured to act as a POSIX-thread library for Linux that can be used to provide concurrency at library level to Ada and C applications. In the case of the Ada language, MaRTE OS is used as the Pthread library that supports Ada tasking for the GNAT compiler. With this configuration, MaRTE applications are executed as standard Linux processes. In the uniprocessor version of MaRTE, a single Linux thread was used to serve all the library-level MaRTE threads required by the user application. This approach has been extended in the multiprocessor version of MaRTE by using one Linux thread for each processor. In this solution Linux threads play the role of processors and Linux signals sent between threads play the role of the IPIs in a real SMP system.

Previously to beginning the OS initialization process, the initial thread of the process assigns itself to the first processor in the system. Afterwards, the initial thread (taking the role of the BSP) performs the first stage of the initialization process: initializes the data structures, unmasks the signals used to emulate the IPIs and creates as many Linux threads as processors are in the system (each thread is assigned to a different processor).

In the second stage, the BSP (the initial Linux thread) sends an IPI via a Linux signal to each AP (the other Linux threads) to execute their respective library-level idle threads. Subsequent IPIs will be used to dispatch the application library-level threads in the different processors.

6. Conclusions

This paper presents the implementation of an SMP Ada platform for two architectures:

- MaRTE MP for XtratuM: this architecture can be used in embedded mixed-criticality systems allowing the execution in the same computer of several multiprocessor real-time Ada applications along with one or more partitions executing a Linux system with soft or no real-time requirements.
- MaRTE MP for Linux: emulation of an SMP system that can be used for educational or research projects and has served us as a perfect test bed for the MaRTE multiprocessor implementation.

The work summarizes the main changes required to adapt the uniprocessor version of MaRTE OS to an SMP architecture. It also describes the small modifications performed in the GNAT run-time library.

The work presented in this paper can be used as a starting point for further research projects:

- The described initialization and dispatching strategies can be used to develop a bare-machine multiprocessor version of MaRTE OS.
- This work can be a starting point to provide a full open real-time implementation of the Ada 2012 SMP support.
- The platform developed can be used as test bed to explore mechanisms to improve the parallelism in the MaRTE OS kernel.

References

[1] Ada Reference Manual. Language and Standard Libraries - International Standard ISO/IEC 8652/2012 (E) with Technical Corrigendum 1 and Amendment 1. (2013)

[2] M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge, "XtratuM a hypervisor for safety critical embedded systems," Proc. of the 11th Real-Time Linux Workshop, Dresden, Germany, (2009).

[3] M. Aldea and M. González. "MaRTE OS: An Ada Kernel for Real-Time Embedded Applications". Proc. of the International Conference on Reliable Software Technologies, Ada-Europe 2001, Leuven, Belgium, in Lecture Notes in Computer Science, LNCS 2043 (2001).

[4] M. Aldea Rivas, M. González Harbour and J. F. Ruiz. "Implementation of the Ada 2005 Task Dispatching Model in MaRTE OS and GNAT". Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe, Brest, France, in Lecture Notes in Computer Science, LNCS 5570 (2009).

[5] M. Aldea Rivas and José F. Ruiz. "Implementation of the new Ada 2005 real-time services in MaRTE OS and GNAT". Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe, Geneva, Switzerland, in Lecture Notes in Computer Science, LNCS 4498 (2007).

[6] H. Pérez Tijero, and J.J. Gutiérrez. "Experience with the integration of distribution middleware into partitioned systems," Proc. of the 17th International Conference on Reliable Software Technologies, Ada-Europe, Berlin, Germany, in Lecture Notes in Computer Science, LNCS 7896 (2013).