Extending FreeRTOS to Support Dynamic and Distributed Mapping in Multiprocessor Systems

G. Abich[1], M. G. Mandelli[2], F. R. Rosa[1], F. Moraes[3], L. Ost[4], and R. Reis[1]

[1]PPGC / PGMICRO Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
[2]Department of Computer Science,University of Brasilia, Brasilia, Brazil
[3]FACIN - PUCRS - Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
[4]Department of Engineering - University of Leicester, Leicester, UK
{*gabich,frdarosa,reis*}*@inf.ufrgs.br*, *mgmandelli@unb.br*, *moraes@pucrs.br*, *luciano.ost@leicester.ac.uk*

***Abstract*—With the ever-increasing complexity of both embedded application workloads and multiprocessor platforms grows the demand for efficient mapping heuristics able of allocating several application workloads at runtime. The majority of promoted mapping techniques are bespoke implementations that consider an in-house operating system, which is developed to a particular architecture, restricting its adoption in other platforms. This work proposes a FreeRTOS extension that supports distributed task mapping heuristics, which enables to balance application workloads in multiprocessor architectures at runtime. Promoted extension is validated through a trustworthy number of scenarios considering large scale Cortex-M-based multiprocessor systems executing up to 600 application tasks.**

***Index Terms*—Dynamic Mapping, Embedded Kernel, Multiprocessor Systems, Modelling and Simulation.**

I. Introduction

Multiprocessor embedded architectures are driven by power wall, performance scalability and reliability challenges. Aiming to effectively scale up system performance while meeting energy-efficiency constraints, multiprocessor architectures are dividing application workloads among multiple threads/tasks [1]. The way such tasks are mapped onto the processing elements (PEs) has a significant impact on system performance, energy-efficiency and reliability [2]. With 1000-processors platforms already available in the embedded community [3], grows the demand for distributed dynamic mapping techniques capable of allocating multi application tasks efficiently.

Mapping techniques have been investigated over the last years, considering different optimization goals (e.g. energy consumption, latency, etc). Most of such mapping techniques are customized implementations, which are developed based on an in-house OS. While providing optimized and efficient means for the mapping techniques, in-house OS based implementations usually are processor-dependent. With the advance of embedded processors, at some point, it will become necessary to port such in-house implementations to a more performant or energy-efficient processor architecture. Underlying porting process is likely to lead to extra design, re-validation and, consequentially a hidden cost that may well be quite high.

The *goal* of this work is to provide a version of FreeRTOS [4] that supports a set of distributed and dynamic task mapping heuristics, which can be easily employed by almost thirty different processor architectures. Due to the non-intrusive and flexible implementation, promoted extensions provide an efficient means not only to use and extend available heuristics but also to integrate new ones.

The main *contributions* of this work are the following:

- adapting FreeRTOS to support dynamic task mapping that can be used for allocating multiple application tasks onto large scale system;
- development of a framework based on OVPsim that unifies software development, debugging capabilities as well as platform configuration and evaluation;
- extensive FreeRTOS extension evaluation by using several scenarios, including multiple real benchmarks and platform models.

The rest of this paper is organized as follows. Section II presents related works in OS-based mapping techniques. Proposed FreeRTOS extensions are described in Section III. Section IV presents and discusses the results. Finally, conclusions are discussed in Section V.

II. Related Works

Researches have been proposing and validating their mapping techniques considering different approaches and platform architectures. Some approaches rely on analytical or simplified executable models [2], which are effective means to propose and compare mapping heuristics. In such approaches either kernel or processor architectures are abstracted away, which may lead to a gap between what is proposed and its adoption in a real platform. Given the increasing complexity of embedded applications, mapping techniques are likely to be implemented based on a kernel/OS. Table I presents the state-of-art in OS-based mapping techniques, targeting multiprocessor platforms. These works are ordered by publication year and classified according to different criteria.

The works presented in [7] and [5] are the only ones to support static mapping. In both works, tasks may be remapped based on a task migration technique, which employs different activities (e.g. context saving and restoring) that are not handled by the task mapping process. Excluding the works proposed by [7] and [6], the mapping control management is

978-1-5090-6113-6/16/$31.00 ©2016 IEEE

TABLE I: State-of-art in OS-based mapping techniques.

Author	Kernel	Footprint	Mapping/Management	Validation Scenarios	Processor architectures	Abstraction Level
Busseuil et al. 2011 [5]	In-house	~60kB	Static/Distributed	6x6 / 3 applications	SecretBlaze	RTL
Ma et al. 2013 [6]	μC-OS II	~24kB	Dynamic/Centralized	3x3 / 1 application	>30 Architectures	RTL
Aguiar et al. 2014 [7]	In-house	~24kB	Static/Centralized	6x5 / 3 applications	MIPS & RiscV	RTL
Mandelli et al. 2015 [8]	In-house	~25kB	Dynamic/Distributed	16x16 / 10 applications	Plasma	Multilevel
This Work	**FreeRTOS**	**~16kB**	**Dynamic/Distributed**	**10x10 / 120 applications**	**>30 Architectures**	**OVPSim**

distributed, which is scalable since more than one processor is responsible for mapping the tasks.

The majority of reviewed works employ RTL platforms described either in VHDL [5] [6] [7] [8] or SystemC [8] to promote their mapping techniques. While VHDL-based platforms were validated through small scenarios (e.g. 6x6 platform executing 3 application [5]), the work in [8] employs a 16x16 NoC-based multiprocessor platform with up to 10 applications. While this work validates distributed and dynamic mapping heuristics, it includes two kernels: one for mapper PEs and another to slave PEs, which leads to extra design efforts.

Beyond the proposed approach, only the work described in [6] uses a highly portable commercial kernel. In this work, each PE is considered as a local cluster of 4 processors, which communicate through a native message passing interface (similar to MPI). This work is mainly devoted to researching the local task allocation, which is defined based on system workload and resource availability. The drawbacks of this work are: first, the mapping control is centralized and adopted heuristic is not efficient for large scale systems; second, the evaluation scenario considers a 3x3 NoC-based platform that executes a single matrix multiplication on each PE.

The *original* contribution of this work is the inclusion of dynamic and distributed task mapping techniques in a market leading RTOS kernel, which eliminates extra design and verification time. Different from the reviewed work, the proposed approach has been validated over different processor architectures (e.g. ARMv6-M and ARMv7-M) considering several and large scenarios.

III. FreeRTOS Extension

FreeRTOS is an open-source real-time operating system widely used in embedded system projects. FreeRTOS has an active development community and it provides a number of functions (e.g. real-time schedulers, memory management, etc) and APIs facilities, which have been validated over 30 different processor architectures. Its small footprint also justifies the choice of using FreeRTOS in this work. Aiming at keeping FreeRTOS modularity and flexibility, its original structure was maintained and promoted extensions were developed to operate in a non-intrusive manner. Underlying extensions were developed targeting NoC-based multiprocessor architectures and they are described as following.

A. NoC Communication

To enable data transfers between communicating tasks and PEs, a MPI-like API was developed. Underlying API includes two communication primitives: *MPI_Send* and *MPI_Receive*, which are used to transfer data and management control packets devoted to inter-task communication and system management. To support the exchange of messages through the NoC, a *Task Manager (TM)* with a Task Management Structure (TMS), a *Communication Buffer (CB)*, and a *Task Location Buffer (TLB)* were incorporated into the FreeRTOS kernel. The TMS contains for each task the local ID, the global application ID, the task relationship ID, and the CB. While CB stores the task outgoing messages, the TLB stores the PE physical address where application tasks are allocated.The CB stores the message whenever a *MPI_Send* is invoked and it suspends the sender task when the CB overflows. *MPI_Receive* blocks the task until the data are available while MPI_Send is nonblocking, unless no buffer space is available. The TM determines the sender processor address by checking the tasks allocation. In this case, there are two possibilities: (*i*) the requested message originates from a task mapped into the same processor, thus the TM retrieves the message data from the local task CB and delivers it; (*ii*) the requested message needs to be fetched in another processor. First, the requesting TM sends to the target TM a service message in order to fetch the message data. The target TM identifies the requesting task ID and removes message from the sender CB. Further, it resumes the task if it is suspended, than it allocates the header descriptor and configures the DMA module with the message to be sent. Finally, the packet is delivered and the requesting task is resumed.

Whenever a task has to communicate with another, a System Call instruction (e.g. SVC, for ARM ISAs) is invoked, in user mode, triggering the System Call Handler that executes the communication primitives. The proposed non-intrusive extension modifies the System Call handler to treat the task requests (e.g. send and receive) in a privilege mode, isolating privileged operations and system resources. Note that each System Call instruction carries an embedded number, which is associated to a given service. These services arguments are stacked into registers *r0-r3*.

Further, for each communication the network interface (NI) triggers the interruption handler (called here as *NI Handler*) to deal with incoming packets. The *NI Handler* manages privileged system services such as message request and delivery. In this work, a packet consists of a six 32 bits-wide flits header followed by the payload. The header contains: the destination address, the payload size (header plus message size), the service request, and three flits to service parameters. An outgoing message first allocates a header descriptor, acquires the receiver physical address from the TLB, and configures the Direct Memory Access (DMA) module. This module uses

the payload address to transfer the information between the local memory to the NoC.

B. Distributed Mapping

To validate promoted FreeRTOS extensions and integrated mapping heuristics, a distributed-memory multiprocessor platform is implemented. Fig. 1 illustrates a 4x4 NoC-based platform, with 2x2 clusters, the application path and the tasks execution with message passing interface.

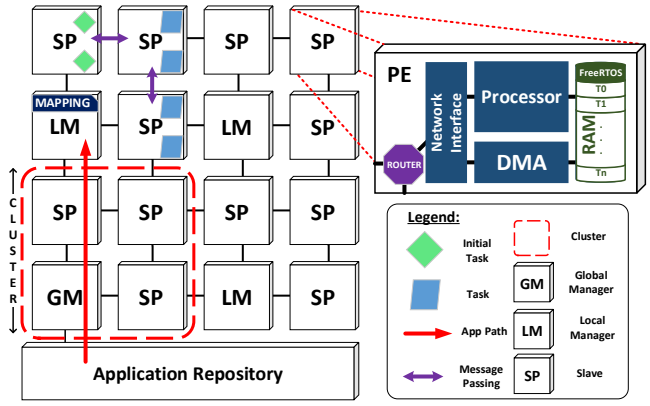

Fig. 1: 4x4 platform instance, partitioned into 2x2 clusters.

The hardware and the software of all PEs are the same. At the system startup, each PE assumes one of the following roles:

- Local Manager PE (LM) - responsible for executing functions such as task mapping within the cluster.
- Global Manager PE (GM) - beyond local management, is responsible for the overall system management, such as defining application-to-cluster mapping, and controlling external devices accesses.
- Slave PE (SP) - responsible for executing application tasks.

Note that the adoption of the same OS at all SPs enables to assign the management task to different PEs. This feature increases the reliability of the system since faults in manager processors do not halt the system. Developed platform is parametrizable, being possible to define: the platform size, the GM position, the cluster size, the maximum number of tasks per PE, the CB size and the application set to execute. The CB size is also parametrizable, allowing to limit the amount of memory to be allocated for inter task communication. The application descriptor is stored in an external memory called application repository which is composed by: application header, tasks header, and precompiled task binary code.

This work deploys two task mapping heuristics: Low Energy Communication based on Dependencies Neighbor (LEC-DN) and Nearest Neighbor (NN) [8]. The heuristic algorithm represents the applications as task graphs based on the its characteristics (e.g load, communication). The application information is loaded to GM, at runtime, from the repository. The LEC-DN heuristic reduces the communication volume through the NoC by nearing communicating tasks that exchange a high communication volume. The initial task mapping evaluates all SPs inside the selected cluster, selecting the SP with the largest number of free SPs around it. The NN heuristic considers only the proximity of an available resource to execute a given task. The search tests all n-hop neighbors, n varying between 1 and the NoC limits in a spiral way, stopping when the first free PE is found.

At system startup, the distributed mapping structure is created according to pre-defined parameters and the NoC communication interface and the scheduler are initialized. At this time, all processors know the GM, the LM, and the cluster limit (all processors within a cluster have this information).

The GM receives from the application repository requests to map new applications. This request contains the number of required resources to execute the application. If the available resources are smaller than the required by the application, the application is scheduled to execute later. The mapping begins with *SearchCluster* function, which analyses the application information and clusters available resources to send the application header to the destination cluster LM. The target LM receives the header and executes a mapping heuristic for the initial tasks. After that, it sends an update TLB message to selected PEs and the GM, and then requests to the GM to send the task code to be allocated in destination SP. The initial tasks are requested to the local manager to map the remaining tasks (same task mapping flow). The destination PE receives the task code and the function *xTaskCreate* creates a new task. Then the required RAM is automatically allocated from the FreeRTOS heap, while the underlying task is included in the list of tasks that are ready to run. At the end of its execution, the task calls *syscall_delete* performing *vTaskDelete*, which removes the task from the scheduler and frees the allocated space. The kernel notifies the LM that the task is finished only after the message buffer is empty. In turn, when the application is finished the LM reports the availability of new resources to GM, which maps other incoming applications or finishes system execution.

IV. Experimental Setup and Results

This work uses the OVPSim [9], an instruction accurate simulator based on dynamic binary translation, to validate promoted extensions. OVPSim supports the simulation of NoC and bus-based multiprocessor platforms. The NoC was implemented using OVPSim APIs (ppm and bhm), and contains a router with five bi-directional ports (input and output data ports), input buffers, and routing and arbiter modules. The NoC deploys a wormhole packet switching mode, XY routing algorithm and distributed arbitration.

In order to validate the proposed work, this Section presents experimental results considering two task mapping heuristics. For this purpose, three applications are used as benchmarks: (*i*) DTW - Digital Time Warping (DTW), with ten tasks; (*ii*) MPEG decoder, with five tasks; (*iii*) DJK - Dijkstra, with six tasks. Table II presents the six evaluated scenarios. Such scenarios use a 10x10 multiprocessor platform instance with a

TABLE II: Evaluated Scenarios (10x10 NoC-based platform).

	Applications	N° of Apps	N° of Tasks
A	120 x MPEG	120	600
B	100 x DJK	100	600
C	15 x DTW, 35 x MPEG	50	325
D	65 x MPEG, 35 x DJK	100	535
E	10 x DTW, 25 x MPEG, 25 x DJK	60	375
F	15 x DTW, 5 x MPEG, 40 x DJK	60	415

5x5 cluster size with different applications. For the evaluated scenarios, we consider a PE with an ARM Cortex-M4F with floating point unit executing two tasks at a time.

The proposed OVP model provides performance metrics, offering to designers low-level performance results. Such metrics are computed at the end of the simulation, and include: *(i)* communication volume; *(ii)* energy spent in the NoC: characterized using the method proposed by [10], calibrated using the ST/IBM CMOS 65 nm technology at 1.0 V, adopting clock-gating, and a 100 MHz clock frequency *(iii)* execution time for each processor: characterized using the model proposed in [11], which counts and captures the executed instructions for a given processor, grouping them according to their behavior.

Table III presents a comparison of the two evaluated heuristics, concerning 2 different metrics: (*i*) communication energy, and (*ii*) execution time. The communication energy presents the cost to transfer the communication volume through the NoC. For this purpose, it considers the distance in hops between each pair of communicating tasks.

TABLE III: Mapping heuristics evaluation.

	Communication Energy (μJ)		Execution Time (1K clock cycles)	
	NN	LEC-DN	NN	LEC-DN
A	16.04	15.74	36493	36806
B	87.90	87.27	36554	37621
C	10.93	10.20	27875	27858
D	39.59	38.95	40036	35554
E	28.42	27.80	32117	31456
F	40.20	39.21	41857	39918

LEC-DN reduces the communication energy for all scenarios compared to NN. This is explained since LEC-DN approximates all communicating tasks and NN approximates only pairs of communicating tasks. The execution time varies according to the scenario, depending on the communication volume, NoC contention, mapping algorithm computation and shared execution of tasks in the system.

Fig. 2 illustrates an example of LEC-DN mapping (scenario C), where each colour corresponds to a different application. Note the locality of the applications, where tasks belonging to the same application are mapped in the same region, with a small distance in terms of number of *hops*.

V. Conclusion

This work proposed a non-intrusive extension to FreeRTOS enabling its adoption in large-scale multiprocessor architectures with distributed-memory organization together with run-time distributed mapping heuristics. Results showed the effectiveness of the included mapping heuristics, which allocated communicating tasks near to each other, reducing the NoC congestion and the communication energy. Additionally, the use of a largely used OS ported to different ISAs facilitates system design and validation, reducing software design cost and time-to-market.

Fig. 2: Application mapping in a 10x10 NoC-based platform.

Acknowledgements

The authors would like to thank Imperas Software Ltd. and Open Virtual Platforms for their support and access to their models and simulator. Fernando Moraes and Ricardo Reis are supported by CNPq. Luciano Ost thanks the support of Santander Universities UK for the travel grant to Brazil.

References

[1] J. Holt, A. Agarwal, S. Brehmer, M. Domeika, P. Griffin, and F. Schirrmeister, "Software Standards for the Multicore Era," *IEEE Micro*, vol. 29, no. 3, pp. 40–51, 2009.

[2] M. H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *DATE*, 2016, pp. 854–857.

[3] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baas, "A 5.8 pJ/Op 115 billion Ops/sec, to 1.78 trillion Ops/sec 32 nm 1000-processor array," in *Symposium on VLSI Circuits*, Jun. 2016.

[4] R. Barry, "FreeRTOS - Real Time Operating System reference manual," 2011. [Online]. Available: http://www.freertos.org/

[5] R. Busseuil, L. Barthe, G. M. Almeida, L. Ost, F. Bruguier, G. Sassatelli, P. Benoit, M. Robert, and L. Torres, "Open-Scale: A Scalable, Open-Source NOC-based MPSoC for Design Space Exploration," in *RECONFIG*, 2011, pp. 357–362.

[6] J. Ma, F. Fu, Z. Liu, Z. Wu, and J. Wang, "μC-OS II-based Operating System design for cluster in NoC-based MPSoC," in *ICSPCC*, 2013.

[7] A. Aguiar, S. J. Filho, F. Magalhaes, and F. Hessel, "On the design space exploration through the Hellfire Framework," *Journal of Systems Architecture*, vol. 60, no. 1, pp. 94–107, 2014.

[8] M. Mandelli, G. Castilhos, G. Sassatelli, L. Ost, and F. G. Moraes, "A distributed energy-aware task mapping to achieve thermal balancing and improve reliability of many-core systems," in *2015 28th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Aug 2015, pp. 1–7.

[9] "OVPsim Simulator," 2016. [Online]. Available: http://www.ovpworld.org/technology_ovpsim

[10] W. Hu, X. Tang, B. Xie, T. Chen, and D. Wang, "An Efficient Power-Aware Optimization for Task Scheduling on NoC-based Many-core System," in *CIT*, 2010, pp. 171–178.

[11] F. Rosa, L. Ost, R. Reis, and G. Sassatelli, "Instruction-driven timing CPU model for efficient embedded software development using OVP," in *ICECS*, 2013, pp. 855–858.