IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 2, 2019, accepted July 22, 2019, date of publication August 27, 2019, date of current version September 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2937730

DRACON: A Dedicated Hardware Infrastructure for Scalable Run-Time Management on Many-Core Systems

DANIEL GREGOREK, JOCHEN RUST, AND ALBERTO GARCIA-ORTIZ
Institute of Electrodynamics and Microelectronics, University of Bremen, 28359 Bremen, Germany

Corresponding author: Daniel Gregorek (gregorek@item.uni-bremen.de)

This work was supported by the Institute of Electrodynamics and Microelectronics, University of Bremen, Germany.

ABSTRACT Many-core architectures integrate a large number of comparatively small processing cores into a single chip. However, the high degree of parallelism increases the run-time resource management complexity and overhead. The employment of dedicated hardware enhancements potentially enables a high quality of the resource management while management overhead is mitigated. To exploit the potential of hardware enhancements, we propose a dedicated infrastructure for run-time resource management on homogeneous MIMD many-core processors. For hardware enhanced resource management, a scalable and cluster-based system architecture is implemented. The resulting architecture (DRACON) utilizes message passing based communication, the dedicated infrastructure and hardware accelerators for resource management. A comprehensive evaluation for DRACON and reference architectures is performed using a transaction level simulation framework and dynamic task management as a use case. As benchmarks, synthetic models and task graph models of real-world applications are applied. The results reveal the limited scalability of classical architectures for resource management on many-cores. It is therefore necessary to apply cluster-based or moderately distributed architectures for many-core resource management. Further, the results demonstrate a significant performance improvement for the DRACON architecture at a number of hundreds of processing cores. Our evaluations show that DRACON generally outperforms software-only run-time management on many-core and achieves a performance improvement of up to 15.21% for single-program and more than 6% for mixed workloads.

INDEX TERMS Computer architecture, many-core, dynamic run-time management, dedicated hardware.

I. INTRODUCTION

The evolution of microprocessors encountered the power wall during the early 2000's [16]. Subsequently, energy efficiency became a mandatory design objective of microprocessors. Therefore, common multi-core processors employ multiple large-sized cores in parallel to improve the performance while maintaining an affordable power level. In contrast, a many-core employs an even larger amount of small-sized cores which is expected to provide a better energy/performance trade-off [4], [7]. Nevertheless, the advancements of CMOS technology reached a fundamental limit of threshold voltage scaling and entered a leakage limited regime [50]. Consequently, microprocessors are expected to face extensive device under-utilization (dark silicon) due to power and application concurrency limitations [19]. The run-time manager (RTM) of a many-core constitutes an essential facility to tackle the challenges of dark silicon [27]. However, the large amount of processing cores and the potentially fine-grained user tasks cause the RTM overhead to rise [34]. Accordingly, innovative and competitive approaches for many-core RTM are required.

The associate editor coordinating the review of this article and approving it for publication was Juan Touriño.

A. MANY-CORE RTM

In the many-core domain, issues like task- and power-management become eminent. Further, pervasive synchronization problems must be addressed for user applications and RTM management tasks as well. Additionally, the requirement of scalability is raised to a many-core RTM. Therefore, a many-core RTM must be designed in a way to provide a maximum of performance independently of the number of available processing cores. It is therefore

necessary to maximize locality and to minimize the contention for data resources [33]. With the rising amount of processing cores, the application of distributed RTMs and private local memory becomes considerable for many-core processors. Another major approach to improve the scalability and to mitigate the contention for resources is the employment of dedicated RTM processing cores.

The Linux OS kernel is a representative for a symmetric software-based run-time system. Linux provides good scalability and high performance, even in the many-core domain [9]. Due to the shared-memory design, Linux depends on high-performance cache coherence and fine grain lock access. Asymmetric approaches like AsymOS dedicate processing cores to dedicated RTM functionality [40]. The approach allows the partitioning of resources between user applications and the RTM. One of their advantages lies in the reduced contention between user and RTM the available resources like processing cores, memory and interconnects. As an example, modifications to the Linux architecture using dedicated RTM cores were proposed to eliminate application interferences due to (non-required) periodic RTM kernel invocations [2].

The factored operating system (fos) dedicates cores for kernel operation to provide a scalable many-core RTM [53]. Every processing core runs a lightweight fos-microkernel while the system services are implemented inside dedicated and distributed system servers. Due to the dedicated system servers, there is no additional overhead for switching between kernel and user mode. The user tasks and the fos system servers communicate by means of message passing. Similarly, the exokernel-like approach Corey also allows dedicated cores to run dedicated kernel functionality [8]. Corey delegates control of shared data structures to the user applications to improve performance and scalability. The Multikernel (Barrelfish) also has a distributed/networked approach but tries to address heterogeneous hardware as well [5]. The Tessellation approach uses space-time partitioning of the available resources for the user applications and system services [13]. The approach explicitly tries to address the issue of Quality-of-Service (QoS), which is important for (soft) real-time systems. In particular, Tesselation explicitly addresses the possibility to exploit hardware enhancements provided by hardware platform. Distributed or agent-based approaches for run-time task mapping like ADAM and DistRM focus on minimization of communication overhead while trying to achieve a mapping quality close to a centralized (global view) approach [21], [32]. Most recently, the distributed operating system nOS plans to address DVFS on a many-core platform [30].

The future of many-core RTM potentially applies asymmetric/dedicated cores and distributed approaches. Key issues to achieve scalability are the maximization of locality and mitigation of the management overhead. Symmetric approaches (e.g. Linux) still provide sufficient scalability, however locking is expected to become a bottleneck in the future. Furthermore, the latency introduced by every software-based RTM potentially limits the employment for many-core systems.

B. HARDWARE ENHANCED RTM

The multitude of run-time management approaches is extended by the possibility of enhancing RTM performance by additional hardware. Especially, the emergence of many-core systems induces a trend towards hardware enhanced run-time management. This trend is caused by certain challenges for many-cores: (1) Many-core can only be leveraged by applications, if they contain sufficient task/thread-level parallelism. Therefore, many-core applications tend to contain more fine-grained parallelism. (2) Locality is getting crucially important. Data availability must be optimized while contention must be minimized. Potentially, the influence of communication versus computation raises. Therefore, the requirements to the RTM scheduling/mapping decisions increase. As a consequence, the impact of the RTM overhead potentially rises: Due to the increasing number of fine grained user tasks and the increasing requirements regarding the management decisions, the management complexity and overhead per tasks can be expected to become larger. One of the directions to address these challenges for many-core run-time management is the enhancement of the RTM by means of dedicated hardware.

The evaluation of typical benchmark applications revealed an average RTM contribution of 9.4%, and for I/O intensive applications a contribution of 97.2% to the system workload [41]. Generally, the management overhead causes contention for computing, memory and communication resources between the user application and the RTM. Therefore, one of the purposes of RTM hardware enhancements is to avoid additional management overhead by means of dedicated hardware [42]. Intrinsic capability of a dedicated and parallel hardware implementation is the possibility to perform RTM operations without interference to the user application. More particular, a hardware implementation can be optimized to exhibit significantly less jitter and lower latency compared to a software approach [1]. Besides performance, the realization of the RTM in hardware has the potential for a higher energy efficiency. These benefits are desirable for emerging many-cores, soft and hard real time environments and many other application scenarios. Drawbacks of a hardware implementation are the additional area and wire overhead and potential restrictions in terms of the RTM portability. One of the primer requirements to a hardware implementation for RTM is the reoccurrence of the RTM operations [49]. However, the decision which parts to implement in hardware instead of software is not trivial. In the many-core regime, the amount of available parallel computing power for user application and for RTM raises, i.e. there may be sufficient computing power to perform an RTM operation in software. Therefore, the dedicated RTM hardware should only deal with time critical operations.

The architecture of a hardware enhanced RTM for many-core may be constituted by means of tightly coupled

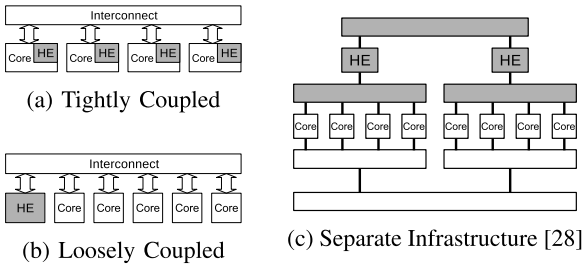


FIGURE 1. Architectural patterns for hardware enhanced RTM. Hardware enhancements are drawn gray.

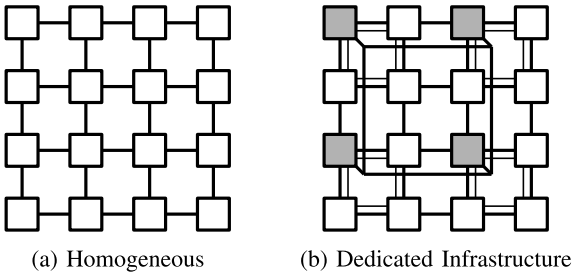


FIGURE 2. Conceptual comparison of (a) homogeneous many-core system versus (b) many-core containing a dedicated run-time management infrastructure. The dedicated infrastructure reserves processing cores for RTM computation (drawn gray) and applies additional interconnects. The number of processing cores is identical for both cases.

modules, loosely coupled modules or a separate infrastructure. Fig. 1 outlines their different architectural patterns. Actual realizations may use a combination or mixture of the presented patterns. Additionally, the architecture of the RTM enhancements depends on the overall system architecture of a many-core, e.g. the memory hierarchy and the interconnect topology.

Depending on the actual realization, different RTM configurations may be assigned to the architectural patterns. As an example, the hardware modules at the loosely coupled approach may operate as a Master module controlling the processing cores, another approach may use a loosely coupled hardware enhancement as a Slave module which is controlled by a software RTM running at the cores. Additionally, depending on e.g. the type of a system call, the same hardware module may operate as a Master in one situation and as a Slave in another. Different functionalities (e.g. scheduling, synchronization, power management) may be assigned to the hardware enhancements. As an example, the STEPNP MP-SoC platform provides dedicated hardware enhancements for message passing, context switching and task scheduling [44]. Also depending on the actual realization, the tightly coupled modules may serve as an enhancement of the core itself, or as an enhancement of the interconnect interface (e.g. a NoC router). A separate infrastructure usually provides dedicated hardware resources for both RTM computation and RTM communication.

C. CONTRIBUTION

DRACON is an experimental many-core architecture providing a dedicated infrastructure for hardware enhanced run-time management. This work extends our contributions presented in [22], [25] with respect to a systematic RTM taxonomy, the integration of distributed RTM hardware accelerators and a comprehensive evaluation of DRACON vs. state-of-the-art HW and SW solutions. Our architecture instantiates multiple distributed hardware modules dedicated to RTM functionality. Due to the distribution of the hardware modules, a high flexibility, scalability and large optimization potential exists. DRACON particularly aims at embedded applications which require hundreds of processing cores, dynamic run-time management, a high responsiveness and low-power operation. However, other requirements or many-core application scenarios may be applied and also benefit from the approach (e.g. the separation of the hardware for user applications and the RTM enhances the overall security capabilities of the systems).

Based on preliminary work it became obvious, that a centralized architecture for hardware enhanced RTM does not provide sufficient scalability for many-core [24]. The instantiation of a single hardware enhanced RTM provides the possibility to highly optimize the hardware module, however it still may become a performance bottleneck. If the number of cores exceeds typical multi-core scales and the task sizes are small, the cores cannot be fully utilized anymore by the centralized RTM. To overcome the limitations of a centralized hardware architecture, DRACON facilitates a clustered and hardware enhanced approach for run-time management. The hardware enhancements constitute a dedicated infrastructure providing dynamic run-time management to the user application. Fig. 2 shows the conceptual comparison of a homogeneous many-core versus a system with a dedicated infrastructure (drawn gray).

The following Section II compares to related work. Section III presents our contributions including the architecture of the DRACON RTM. Section IV evaluates the performance of DRACON including a comparison to reference RTM implementations. Finally, Section V concludes the paper.

II. RELATED WORK

The contribution is compared to related work regarding the scalability of the RTM architecture and regarding the scope of the employed RTM hardware enhancements. Generally, it is expected that a larger scope of the RTM hardware enhancements increases the performance but exhibits a higher area overhead.

A. ENHANCEMENTS FOR SYNCHRONIZATION

The performance of synchronization mechanisms is of major importance in parallel computing. Due to the increasing number of processing cores and the increasing number of fine-grained user tasks the synchronization effort is expected to rise. Therefore, a multitude of synchronization approaches implemented in hardware and software exist. The following section outlines state-of-the-art approaches of hardware enhancements for task synchronization.

STHORM is a clustered many-core computing platform [6]. The platform follows a globally asynchronous locally synchronous (GALS) approach and targets embedded applications, e.g. in the field of computer vision. It consists of up to 4 cluster each of which contains 1-16 processing cores. Additionally, each cluster contains a dedicated module for hardware synchronization (HWS) and a cluster controller [51]. The clusters can be further extended by hardware processing elements (HWPE) as accelerators for user computation. STHORM potentially shares the global interconnect between user and RTM. The programmable processing cores are inside the so-called ENCore 16 block. The most peculiar components of the HWS are atomic counters, a programmable notifier and a dynamic task allocator. The components of the HWS allow to provide a binary mutex and interrupt-based event notification (no busy-wait). As an example, the task synchronization capabilities of the HWS module are exploited by the hardware-assisted run-time software (HARS). However, until today, HARS only provides intra-cluster task synchronization [36].

Further, a low power heterogeneous many-core SoC for multimedia applications is presented by the Toshiba corporation [38]. The SoC contains two ARM cores, two reconfigurable processors, hardware accelerators and two many-core clusters, each cluster containing 32 processing cores. A bus interconnect is used for communication between the many-core clusters. A single many-core cluster consists of 32 VLIW processors called multimedia processors (MPB), a NoC connecting the processors, local caches and a cluster control module. For the NoC a tree topology was selected. The cluster control module contains hardware semaphores which assists in the synchronization of the VLIW processing cores.

The MPPA-256 is a clustered and NoC-based many-core architecture [17]. The MPPA clusters are connected by means of data and control NoCs. Similar to STHORM, each of the 16 processing clusters contains a dedicated and full-fledged system core for resource management (RM). Basic purpose of the RM system cores is the NoC RX event management and DMA activation [18]. Additionally, the MPPA contains four I/O subsystems, each of which are controlled by a quad-core system processor. In particular, the main processes of the applications reside at the dedicated quad-core system processor of the I/O subsystem. Therefore, the computation on the clusters is initiated by the I/O subsystem.

B. ENHANCEMENTS FOR SCHEDULING

Hardware enhancements provide a promising approach to tackle the complexity of the scheduling problem. Particularly for a large number of cores and small sized tasks, the rising impact of the scheduling overhead demands for a dedicated hardware implementation [34]. Therefore, hardware enhancements for scheduling are especially considerable for multi- and many-core systems. Additionally, hardware schedulers also provide worthwhile characteristics for real-time systems. As an example, the hardware implementation of a priority scheduler offers substantial speed-up opportunities compared to a software-based implementation. Generally, the dedicated implementation of a hardware scheduler provides an additional space-time trade-off regarding the amount and time-multiplexing of instantiated data path components. As an example, the sorting procedure to dynamically obtain the highest priority task can be accelerated by applying multiple hardware comparators. The following section presents an outline of state-of-the-art approaches in hardware enhancements for scheduling with a focus on their architectural properties and implementation details.

Park *et al.* [43] implemented a hardware operating system kernel (HOSK) as a loosely-coupled centralized master module. Additionally to a common multiprocessor interconnect, they apply a dedicated interconnect for context management. The HOSK supports hardware implemented thread scheduling, management of inter-process communication and context switching in a multicore environment. The HOSK thread scheduler applies dedicated RAM memory for storing thread control data (instead of registers).

Nexus++ is using a Master-Slave configuration for task dependency resolution and scheduling at run-time [15]. It supports a task-based programming language using pragmas. Origin of Nexus++ is a scalability bottleneck in the software-based run-time library for task management. Peculiarity of Nexus++ is the high degree of hardware implementation, i.e. it is a fully hardware implemented task management module. Recently, Nexus++ has been extended by using multiple parallel units in the data path for task management [14]. Similarly, Task Superscalar proposes an out-of-order pipeline for resolving task dependencies at run-time [20]. The approach is inspired by the classical out-of-order pipeline to exploit instruction-level parallelism. Task Superscalar targets for task management on many-core and uses a distributed front-end, but a single centralized queue for task dispatching. Nexus++ and Task Superscalar facilitate a hardware enhanced centralized RTM architecture for task management. However, both solutions lack scalability due to the drawbacks of centralism in computation and communication.

IsoNet targets a scalable and distributed architecture for task queueing and load balancing [35]. IsoNet has been inspired by Carbon which uses a centralized hardware task queue [34]. In contrast, IsoNet comprises a dedicated hardware mesh network (separate infrastructure) to perform the task queueing and task migration at run-time. However, the actually implemented protocol to perform the load balancing is based on a tree structure comprising a (centralized) root node. The reason is probably due to the lower complexity of the tree based protocol which is fully implemented in dedicated hardware. The approach claims to be scalable for more than 1024 cores, but is constrained to independent tasks. Also, due to a limited synchronization scheme, the Isonet nodes may not find a globally optimal load balance.

As an alternative to fully dedicated networks, the interfaces of existing interconnects may be enhanced by dedicated RTM

functionality. The Eclipse architecture employs a dedicated hardware shell at the interconnect interfaces which provides distributed task scheduling [46]. Heisswolf et al. propose a hardware extension to NoC routers to provide quality-of-service to the applications by means of decentralized NoC region management [26]. Similarly, the routers might be extended to accelerate the application mapping in a decentralized fashion [52]. Also, the routers could be enwrapped by additional hardware to dynamically map the task communication to the common electrical or an additional optical NoC [10], [11].

In comparison, the clustered DRACON architecture proposes a dedicated infrastructure for RTM. The infrastructure comprises RTM computation and communication for scalable, flexible and global-view run-time resource management.

III. DRACON APPROACH

Distributed architectures have better scaling properties compared to centralized management architectures. However, distribution comes with the overhead of data replication and communication. Therefore, DRACON endeavours a clustered RTM architecture to present a suitable trade-off between a centralized and a fully distributed system.

A. RTM TAXONOMY

This work contributes a particular taxonomy to describe and compare actual RTM architectures. The taxonomy uses three major RTM components to assemble an RTM: RTM master, RTM slave and RTM data table. An RTM master performs decision making and global view operation. An RTM slave assists locally and provides the interface to the processing cores. An RTM data table contains the management data.

The taxonomy allows to represent the *logical* architecture of an RTM. It describes the interaction, the responsibilities and the hierarchy of the RTM components. Actually, different logical architectures can be mapped onto the same hardware architecture (e.g. a homogeneous many-core may run various RTMs). Therefore, the taxonomy is independent of the underlying hardware architecture. Accordingly, the employment of dedicated resources (see Sec. I-B) is only defined by the mapping of the logical RTM architecture to the hardware. As an example for using the taxonomy, Fig. 3 outlines certain logical RTM architectures. The related work Nexus++ corresponds to a centralized architecture (Fig. 3a), fos and STHORM (and DRACON) correspond to a clustered architecture (Fig. 3b), Linux to a symmetric architecture (Fig. 3c) and Isonet to a distributed one (Fig. 3d).

B. HARDWARE ARCHITECTURE

The DRACON approach facilitates a dedicated infrastructure for run-time management [22]. The overall RTM system constitutes a hardware enhanced and clustered RTM architecture. The approach is particularly inspired by the Nexus++ hardware task manager [15], the software-based factored operating system (fos) [53] and the STHORM platform [39].

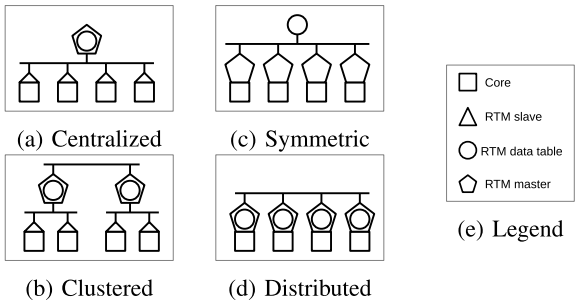


FIGURE 3. Examples of RTM architectures. A centralized architecture 3a has a single RTM master. A clustered architecture 3b groups the computation resources while each group is controlled by one RTM master. A symmetric architecture 3c has one RTM master per core and typically employs shared memory. A fully distributed architecture 3d has one RTM master per core each having private data. Subfigure 3e shows the legend.

The dedicated infrastructure enhances a common many-core system and is hierarchical due to the clustering of the resources to be managed. It is also distributed, since each cluster can be managed autonomously. The RTM architecture shown in Fig. 3b is established by an infrastructure of dedicated hardware resources enhancing a baseline many-core. The baseline many-core system consists of numerous processing cores, connected by a common many-core interconnect. As a low-latency interface, a closely coupled hardware enhanced RTM slave (HSLV) is connected to each core. The HSLVs are connected to a local management interconnect, which constitutes a cluster of core + HSLV pairs. Each cluster is controlled by a hardware enhanced RTM master (HMST). Communication between the HMSTs is done via a global management interconnect. Fig. 4 gives an outline of the proposed system architecture.

It is the exclusive purpose of the dedicated infrastructure to perform the RTM computation and communication. Therefore, no user computation or communication is allowed at the dedicated management infrastructure. Oppositely, the baseline system is not necessarily reserved for the user computation and communication. Therefore, the baseline system could also be used by the RTM adaptively at run-time. In the current contribution, run-time adaptivity is not used, i.e. exactly one RTM master is instantiated per HMST and the RTM runs only at the dedicated infrastructure.

1) BASELINE SYSTEM

For the elaboration of the DRACON approach, a homogeneous many-core is considered as an initial baseline system. The many-core may consist of arbitrary processing cores and local memories which are connected by a common many-core interconnect. Also, the DRACON approach does not rely on any particular topology of the interconnect. Therefore, mesh, torus or other interconnect topologies may be applied. By default, DRACON assumes a non-coherent memory sub-system (similar to Intel SCC [31] and Swallow [29]). The baseline system is not necessarily clustered by itself, the hierarchy may only be constituted by the RTM dedicated infrastructure.

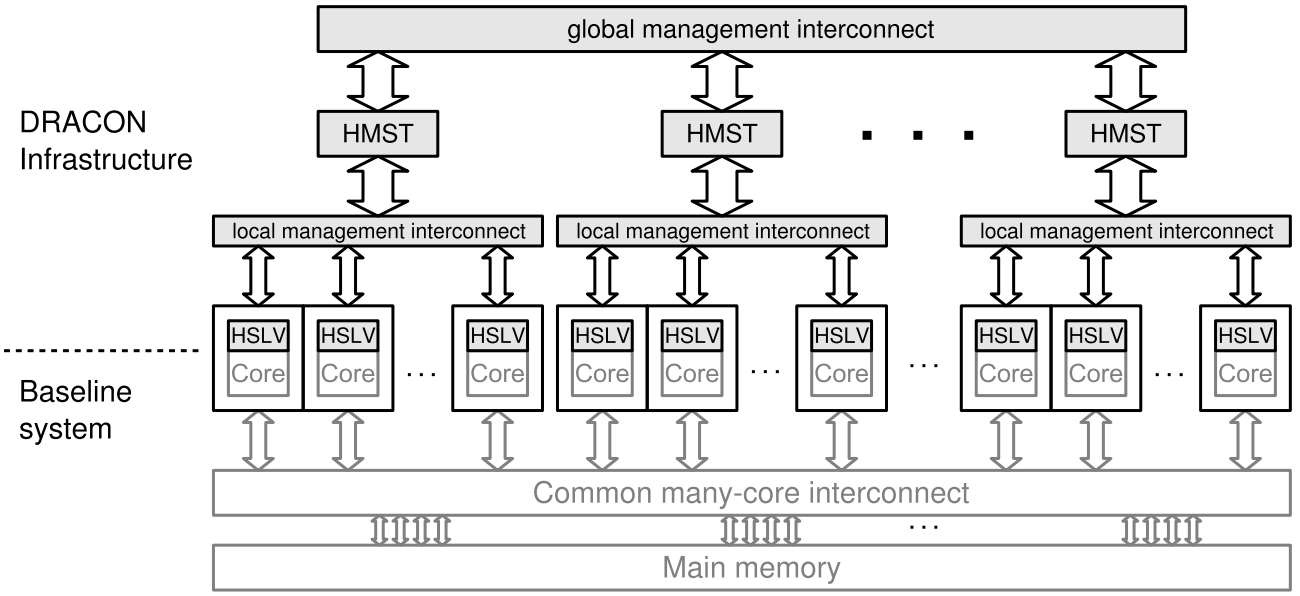


FIGURE 4. Architecture of the DRACON infrastructure for run-time management (drawn gray) on top of a baseline many-core system (drawn dimmed). The DRACON infrastructure consists of hardware enhanced RTM Masters (HMST) and hardware enhanced RTM Slaves (HSLV). The RTM components communicate by means of message passing and perform the management operations at run-time. The processing cores are connected by a common many-core interconnect.

The primary duty of the processing cores is the execution of user tasks. The cores therefore consist of a programmable processor, local memory and a network interface. The common baseline interconnect is responsible for data transfer between the cores (task communication). It further allows to have multiple connections/ports to the main memory. The baseline interconnect may connect to any external memory or I/O. However, off-chip communication is out of the scope of this work.

2) DEDICATED INFRASTRUCTURE

The DRACON approach enhances a common many-core architecture by means of a dedicated RTM infrastructure. In a default case, the baseline interconnect is assumed to be a NoC having a mesh topology. Therefore, the placement/positioning of the hardware enhanced RTM masters is expected to have an effect on the locality, i.e. the cores which are grouped into a cluster should have a small Manhattan Distance at the baseline many-core interconnect. Therefore the chip is split into rectangles and the RTM masters are placed evenly all-over the chip. Fig. 5 shows an XY view of the placement of RTM masters for an 6x8 configuration having 4 HMSTs and 44 cores.

a: HW-ENHANCED RTM MASTER

Each instance of a hardware enhanced RTM master (HMST) realizes one instance of an RTM master (see III-A) on a dedicated hardware resource. Each HMST has its private address-space, the communication between one HMST and it's HSLVs and between the HMSTs is strictly message based. An example message protocol for task management is presented in Sec. III-D. For RTM communication, an HMST must contain an interface to the dedicated management interconnects. To execute the actual RTM master, a programmable RISC processor and SRAM memory may be applied. To speed-up RTM operations, a HMST can be enhanced by additional hardware accelerators. In our present scenario we consider HW modules for the acceleration of task synchronization, scheduling and mapping (see also Sec. III-E). Also, further hardware optimizations can be seamlessly integrated, e.g. an optimized instructions set for the programmable RISC processor.

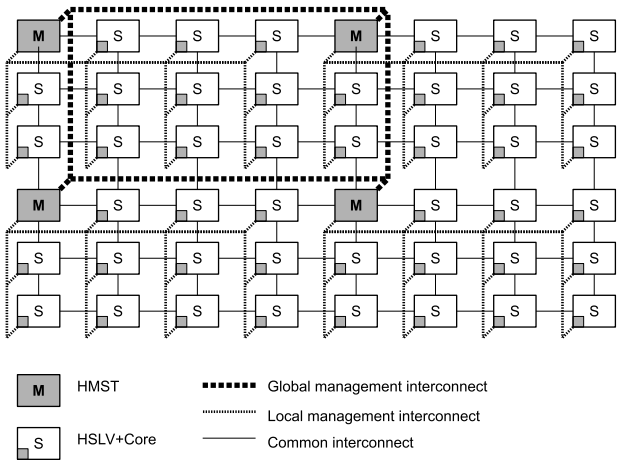


FIGURE 5. Placement of hardware enhanced RTM components (XY view).

b: HW-ENHANCED RTM SLAVE

The hardware enhanced RTM slaves (HSLV) consist of a messaging interface, a system-call dispatcher and a tightly-coupled interface to the core. Also, task queues and

TABLE 1. Overview of DRACON system calls with inputs and outputs.

Name	in	out	Description
`sysc-task-spawn`	`imem`, `dmem`, `cnt`	`bool`	Recursively spawn number of child tasks (`cnt`) with given instruction- (`imem`) and data-memory (`dmem`) address.
`sysc-task-exit`	`addr`	`bool`	Send a signal to the synchronization barrier given by `addr` and terminate recursive child task. The parent task is restarted, if all child tasks have finished.
`sysc-info-send`	`key`, `value`	`bool`	Send 32-Bit value to key.
`sysc-info-recv`	`key`, `cnt`	`value(s)`	Receive 32-Bit value(s) by key. Blocks the task until all values are available.
`sysc-get-core`		`core-id`	Get numeric identifier of core

multiple context buffers can be integrated. The dedicated hardware HSLV can be implemented with low area overhead and operates in parallel to the core [22]. Any system-call from a user task is fetched by the HSLV and dispatched to its RTM master by means of a dedicated message. Due to the dedicated infrastructure for the run-time management the core is relieved from the execution of the RTM service.

C. TASK MANAGEMENT

Dynamic hardware faults or changes of the user requirements make dynamic run-time task management a necessary feature of a many-core RTM. As a use case for DRACON, a dynamic and scalable run-time task manager is proposed and implemented based on the presented RTM taxonomy (see Sec. III-A). The DRACON task manager constitutes a clustered RTM architecture and provides the synchronization, communication and scheduling of user tasks. The applied task management mechanisms and algorithms are selected based on their practicability to be implemented in hardware.

Due to its generality, the task management follows a rigorous task graph based view to the user application. Every task is fully exposed and handled by the RTM. The implemented task manager does not require a priori knowledge of the characteristics of an application, i.e. the task graph must not be known to the task manager in advance. In particular, the task synchronization and communication is dynamically signalized by the user tasks at run-time. Nevertheless, each task is usually associated to the vertex of a task graph. To optimize the RTM operation at compile- or run-time, the task manager may analyze and exploit the particular task graph properties.

According to the task graph model, we divide the life-time of a task into the phases *receive*, *compute* and *transmit*. Fig. 6 shows an example task graph which will be used throughout the section to highlight the characteristics of the proposed task manager. The task n_c receives its input from the tasks n_{x1}, n_{x2} and sends its output to the tasks n_{y1}, n_{y2}. An identifier (key) is assigned to each edge of the graph (e.g. $k1$ for the edge from n_{x1} to n_c). The task manager provides a set of system calls to the user tasks to realize the dynamic execution of the task graph model. Tasks can be created by using the system call `sysc-task-spawn`. The tasks signalize their termination by means of the system call `sysc-task-exit`. To allow the MPI-like inter-task communication, the task manager provides the system calls `sysc-info-send` and `sysc-info-recv`. An overview of the most important system-calls, which are provided by the RTM is listed in Tab. 1. In our current implementation we assume the system calls to be fully exposed to the user application. However, a lightweight software layer can be applied as future work to enable compatibility to state-of-the-art software frameworks (e.g. OpenMP).

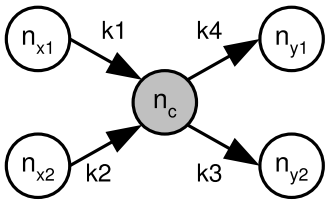
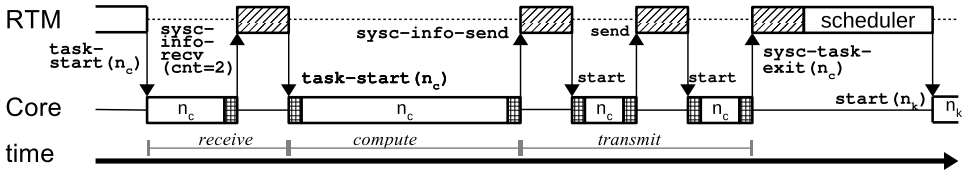


FIGURE 6. Example task graph: The task n_c receives its input data from the tasks n_{x1} and n_{x2} and sends its output to the tasks n_{y1} and n_{y2}.

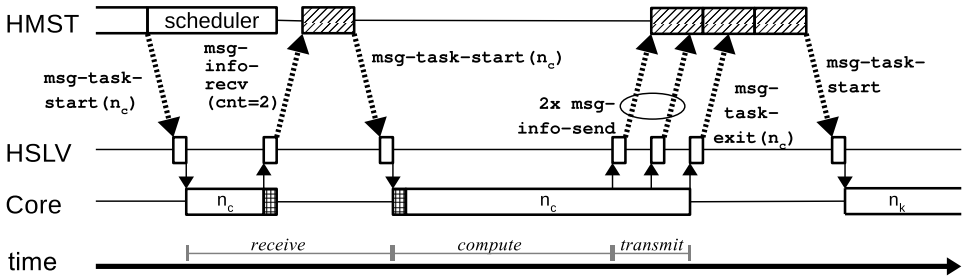
The run-time task management requires computation time to service the system-calls and in particular for the scheduling and mapping of the user tasks to processing cores. As user applications may consist of hundreds or thousands of tasks, a crucial trade-off between the quality and the overhead of the run-time decisions has to be considered. A hardware enhanced run-time manager hides the additional computation time due to the provided parallelism. Fig. 7 illustrates the benefits by means of a comparison between a software-based implementation and the DRACON approach. A software implementation will, at least initially, service a system call at the local processing core and requires more context switches (Fig. 7a). The hardware enhanced DRACON task manager forwards a system call (Fig. 7b) by means of a message to the HMST. The HMST services the system calls and may precompute the scheduling before going idle. If a core becomes idle, the next tasks can be dispatched immediately. The responsiveness of the management system is therefore improved and the cores are given more time for processing the user task.

1) TASK MAPPING AND SCHEDULING

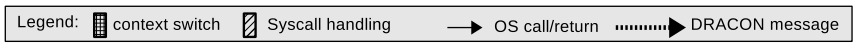
The quality of the mapping and scheduling decisions has significant impact on the overall system performance [47].



(a) Software based RTM



(b) DRACON



(c) Legend

FIGURE 7. Comparison of task management for the task n_c having two input and two output edges. The software based approach (Fig. 7a) runs at the same core and interrupts the user task for RTM service. The software approach schedules serially to the core. The hardware enhanced solution DRACON (Fig. 7b) implements the same RTM interface but sends messages between the HSLV and the HMST. The hardware approach requires less interruption of the user task and schedules in parallel to the core. The outcome is an earlier dispatching of the next waiting task n_k.

However, a timing overhead may be induced when computing the task management at run-time. The possibility of integrating hardware accelerators into the DRACON architecture enables the employment of high quality algorithms with a low timing overhead.

To improve the quality of the mapping, the RTM may leverage run-time information like Manhattan Distance and task communication volume. For scheduling, the RTM can consider the expected task execution times for priority sorting. Due to the clustered architecture, DRACON uses a hierarchical two-step task mapping approach. The first step maps a user task to a cluster, the second step maps to the local core. Task migration between clusters is performed by a load balancing mechanism. Also, the task scheduling operates in a decentralized manner: Each RTM master can operate on its cluster independently from the others. By default, the scheduling is quasi-preemptive: a running user task may be preempted by a higher priority system task but not by another user task.

2) TASK SYNCHRONIZATION AND COMMUNICATION

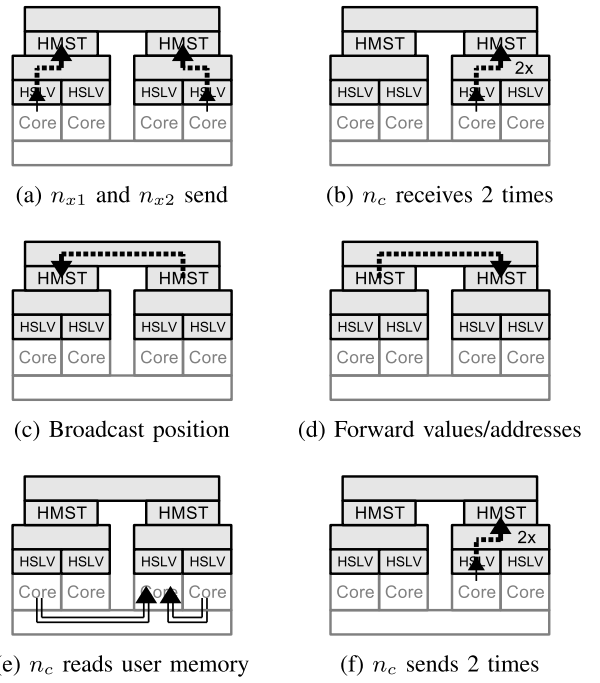
To have a flexible task synchronization and communication sub-system which is able to work in a distributed environment, a communication infrastructure based on a distributed dictionary was developed. Each RTM master maintains a copy of the dictionary and forwards the synchronization messages between the tasks. To speed-up the synchronization, the dedicated interconnects are used for transporting the synchronization messages.

A unique key is assigned to each task communication (i.e. task graph edge). Inside a cluster, user tasks can directly send and receive (non-empty) values by means of the communication key. To handle the communication between tasks residing in different clusters, the sub-system requires knowledge of the locality of the tasks. Therefore, the synchronization sub-system utilizes a dynamic *localize-at-receive* mechanism: The position (cluster) of the key is connected to the position of the receiving task. Every time a read request occurs, and the belonging position of the key is not yet revealed or has changed, the position of the key is broadcasted by the HMST via the global management interconnect. The other HMSTs read that broadcast and update their key positions accordingly. The broadcast is fully transparent to the user tasks, and allows their synchronization, even when tasks change their position from one cluster to another. Any value which is send to a key, who's position is not yet discovered, is pending until the required broadcast occurs. As soon as the position of a given key is known, a transport of the waiting values to the discovered position is triggered.

The task synchronization sub-system can be used to quickly transport memory addresses between communicat-

TABLE 2. Overview of DRACON messages.

Name	Direction	Description
`msg-task-start`	HMST to HSLV	Start task at destination HSLV by means of the task descriptor (`tskd`), stack pointer (`stack`). The message may contain additional data words
`msg-task-spawn`	HSLV to HMST	Spawn new tasks of count `cnt` with given parent, instruction- and data memory address
`msg-task-pull`	HMST to HMST	Request migration of a tasks. The message is used to implement the load balancing mechanism
`msg-task-push`	HMST to HMST	Request start of a task at another HMST. The parameter `rcsv` is required to implement the recursive start-up mechanism
`msg-task-exit`	HSLV to HMST, HMST to HMST	Terminate task and/or signalize to join barrier `join`
`msg-beacon`	HMST to HMST	Broadcast load status
`msg-info-recv`	HSLV to HMST	Let task receive `cnt` synchronization values for `key`.
`msg-info-send`	HSLV to HMST, HMST to HMST	Send synchronization value to key
`msg-info-cast`	HMST to HMST	Broadcast key position



(a) n_{x1} and n_{x2} send (b) n_c receives 2 times

(c) Broadcast position (d) Forward values/addresses

(e) n_c reads user memory (f) n_c sends 2 times

FIGURE 8. Task synchronization and communication example. Task n_c receives two memory addresses by means of system calls (→) and messages via the dedicated interconnects (⇢). Subsequently, the actual data is read via the common interconnect (⇒).

ing tasks. The actual large-volume user data is subsequently transported via the common interconnect. Therefore, arbitrary communication between dynamically mapped user tasks can be resembled. Fig. 8 shows an example sequence based on the task graph in Fig. 6 demonstrating the implemented task synchronization and communication sub-system.

3) APPLICATION START-UP

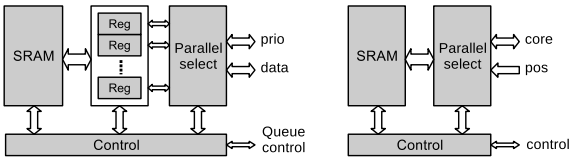
Since the targeted applications potentially have a large number of parallel tasks, the DRACON RTM uses a recursive task fork/join strategy for application start-up [22]. Every recursive task spawns two additional system tasks and then blocks until its child's have terminated. As the last step of the expansion, the actual child tasks of the application are spawned. The final number of working child tasks is fixed and determined by the application profile.

D. DEDICATED MESSAGE PROTOCOL

DRACON sends messages via the dedicated RTM interconnects to implement the communication between the RTM hardware components. Each message has a header and one or more 32-Bit data fields. The header contains the message type, at least the source address, the message priority and a broadcast flag. E.g. load balancing messages may run at a lower priority than task spawn or exit messages. Additionally, a message drop mechanism may be applied to drop certain low-priority messages during high workload on the interconnects. The size of the message header depends on the actual hardware configuration (i.e. number of components/address-width) and the direction of the message. Tab. 2 gives a short outline of the implemented messages. If a message is related to a system call, the HSLV copies the required fields from the registers of the processing core into the message (e.g. key and data value for `msg-info-send`) and dispatches it to the HMST. Furthermore, the potential response from the HMST is injected into the processing core registers.

E. RT-LEVEL IMPLEMENTATION

The essence of the DRACON architecture has been implemented at the register transfer level. The RT level design comprises VHDL models for the dedicated RTM infrastructure and the baseline hardware (see Sec. III-B). Additionally, a low-level C library was implemented as a software interface to the dedicated hardware. In the current implementation, the low-level C library basically acts as an interface between the dedicated message passing infrastructure and the task manager presented in Sec. III-C. But in general, arbitrary hardware/software implementations for many-core RTM can be applied. As an example, a hardware priority queue can be connected to a software programmable RTM processor by means of the C library. Key aspects of the dedicated RTM infrastructure are the hardware enhanced message passing for RTM communication and hardware enhancements for RTM computation. For communication, dedicated RTM modules for message transmission (TX) and reception (RX) are implemented. The implementation of the dedicated management interconnect is loosely based on the AMBA 2.0 specifications [3]. For RTM computation, VHDL modules for the



(a) HW for scheduling (b) HW for mapping

FIGURE 9. Hardware accelerators for dynamic task management.

HMST and HSLV were implemented. The VHDL model for the baseline hardware comprises the processing core and a VHDL behavioral model for accessing shared memory.

The main component of a hardware enhanced RTM master is a programmable PLASMA/mLite RISC processor [45]. The current implementation uses a memory mapped connection scheme to the processor. Therefore, SRAM memories, message TX/RX modules or further hardware accelerators can be connected to the processor seamlessly. The low-level C library is used to access the connected hardware components. The HMST regularly polls for incoming RTM messages. If a message arrives, the corresponding action is performed (see Tab. 1). The hardware enhanced RTM slaves are tightly coupled to custom made processing cores. The HSLV is basically constituted by a finite state machine (FSM) which processes the RTM communication messages and manipulates the register file of the processing core. If a user task raises a system-call, the core activates the FSM by means of the `sysc` signal. The FSM may halt the core to read or write the register file. Also, the FSM may take over the memory interface of the core to exchange data between the register file and the main memory (e.g. to perform a context switch). The core facilitates a sub-set of the MIPS instruction set to model the behavior of the actual user tasks.

As use cases for hardware accelerators, hardware modules for task synchronization, priority scheduling and mapping have been implemented at the RT level. To enable scalability, the module for priority scheduling is constituted by a fully parallel task sorting logic and an additional sequential SRAM memory. Therefore, the highest priority tasks always reside in a parallel region implemented by registers while the lower priority tasks are stored in the sequential SRAM. The architecture of the scalable hardware priority queue is shown in Fig. 9a. The mapping module linearly searches an SRAM which holds the status information of the cores. One SRAM word holds the information regarding 4 cores. During the search, the most suitable core corresponding to a reference position is selected. Fig. 9b shows the architecture of a hardware task mapping module. Additionally, a hardware synchronization module consisting of an array of FIFOs (one FIFO per communication key) was implemented to partly realize the task synchronization sub-system in hardware. The FIFOs either hold the communication values or the tasks waiting for values.

IV. EVALUATION

The following section covers the evaluation of the DRACON architecture compared to particular RTM reference architectures. Sec. IV-A introduces the experimental setup and the properties of the applied RTM reference architectures. Sec. IV-B applies synthetic benchmarks to analyze certain design aspects under particularly adapted conditions. Sec. IV-C evaluates the RTM architectures under realistic conditions using the MCSL benchmark suite. Finally, in Sec. IV-D the gate level area overhead of DRACON is approximated.

A. EXPERIMENTAL SETUP

The evaluation is performed by means of the transaction-level simulation framework Agamid [23]. The framework provides a fine-tuned combination of different levels of abstraction to resemble a task-accurate simulation model. To speed up simulation, task behaviour is abstracted by task execution time and communication volume. However, major emphasis is put on the synchronization model to keep a high degree of accuracy. Further, a highly generic and template based RTM is integrated into the framework to evaluate RTM implementations on a uniform basis. We use Agamid to model HW/SW RTM architectures and compare DRACON to reference models. The timing model is calibrated by means of cycle-accurate evaluations of the RTL model (timing parameters are given in [23, Tab. 3]). By convention, the model is not related to a specific clock frequency and any timing values are given without a unit. All RTM configurations use a quasi non-preemptive scheduling. User tasks may only be preempted by system tasks. The mapping of a user task to a core is only performed, if there is an idle core available. Depending on the RTM configuration, the processing cores are either used for user computation, for RTM or shared between both. All RTM configurations are multi-master systems by default. The multi-master RTMs use a recursive task fork mechanism for application start-up (see III-C.3).

To compare the performance results of the hardware enhanced run-time management, two major reference architectures were implemented. The first one is a classical symmetric software RTM (SW-Symm) using a single shared memory. An aggressive synchronization model is used to simulate access protection to shared RTM resources. The synchronization model uses a semaphore-based locking mechanism and only considers the semaphore access itself as a critical section while the actual read and write operations on the shared resource can happen concurrently. Therefore, the synchronization model mimics a high degree of parallelism due to local caches but guarantees protection of shared resources. The second major reference architecture is an asymmetric software RTM (SW-Asym) having a more progressive distributed architecture. The RTM masters of SW-Asym communicate by means of message passing via a shared interconnect and therefore RTM communication interferes with the user communication. To evaluate the impact of hardware accelerators for task management we distinguish between the baseline DRACON architecture without accelerators and DRACON-AC including accelerators. In addition to the RTM architectures we compare

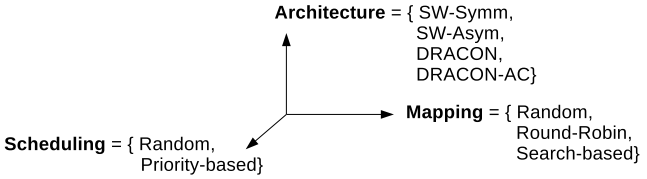
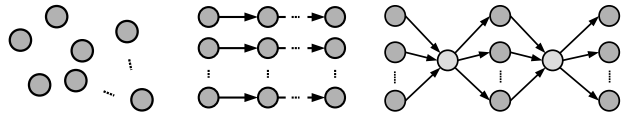


FIGURE 10. Overview of RTM design space parameters.



(a) Independent (b) Horizontal (c) Barrier

FIGURE 11. Synthetic benchmarks.

between different task scheduling and task mapping algorithms. Fig. 10 gives an overview of the RTM parameters which are considered for evaluation.

B. SYNTHETIC BENCHMARKS

In the following section synthetic benchmarks are used to evaluate the performance of the RTM implementations. The purpose of the synthetic benchmark applications is to provide a clear knowledge of the application characteristics and to introduce particular challenges to selected sub-systems of the RTM. In other words, the characteristics of the synthetic benchmarks shall avoid undesired RTM side-effects which would be introduced by real-world benchmark applications. In particular benchmarks consisting of independent tasks, horizontal task dependencies, and barrier dependencies are applied. An overview of the different task graph structures is given in Fig. 11.

Independent tasks (Fig. 11a) are used to analyze the RTM performance for task scheduling and task dispatching. The horizontal benchmark (Fig. 11b) consists of multiple sequences of tasks. Inside a sequence, tasks read input from the preceding tasks and communicate the output to their successor. The horizontal task communication is used to analyze the RTM mapping capabilities. To analyze the task synchronization capabilities of the RTM, a barrier scheme is applied (Fig. 11c). To model a barrier, additional synchronization tasks are inserted into a task graph.

1) SCHEDULING

The following section considers the employment of basic run-time algorithms for scheduling of independent tasks (Fig. 11a) on a single cluster of cores. It is to be investigated, if the employment of a higher quality run-time scheduler (e.g. priority-based) reveals a better result compared to other scheduling algorithms. Thereby, the impact of the actual run-time scheduling overhead is considered. To evaluate the scheduling effects, a synthetic benchmark consisting of independent tasks is considered. The execution time of the tasks is exposed to the scheduler and is taken into account for priority-based scheduling. Tasks with a larger execution time are selected first. Random scheduling is employed as reference and used to normalize the evaluation results. The task sizes l are varied by using a uniform distribution between 1 to 20000. Further, the number of cores and the number of tasks is varied. Each of the design points is tested for 100 iterations, each iteration having a different task set. An algorithm gain G is computed by comparing the speedup using priority scheduling versus random scheduling (see Eqn. 1).

$$G = S_{prio}/S_{rand} = T_{rand}/T_{prio} \tag{1}$$

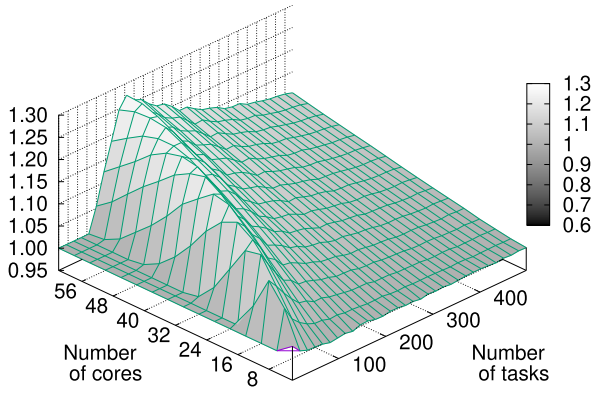
Fig. 12a shows the gain G_{real} by comparing priority scheduling vs. random scheduling using DRACON-AC. The priority scheduling exhibits a performance gain vs. random scheduling of up to 25%. The ratio between SW-Symm and DRACON-AC is shown in Fig. 12b. DRACON-AC using a hardware priority queue exhibits a 5% performance gain considering real RTM overhead for a rising number of tasks. The symmetric software-based RTM allows to use priority scheduling, however it is potentially not scalable if the number of cores becomes larger. The priority scheduling appears generally not applicable for asymmetric software-based RTMs due to the limited number of RTM masters and the high management overhead. For the DRACON architecture, the overhead of priority scheduling can be significantly mitigated by the integration of hardware implemented priority queues (DRACON-AC).

2) TASK MAPPING

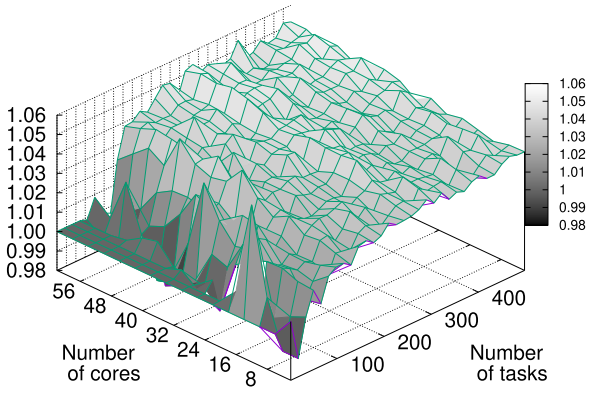
To evaluate the impact of task mapping a synthetic benchmark having horizontal dependencies between communicating tasks is used (Fig. 11b). The amount of sequences is expressed by the variable p. Since each sequence is running independently of the others, p also denotes the degree of parallelism of the benchmark. The optimal mapping of the benchmark would use $m = p$ cores, where each sequence of tasks is mapped onto one of the cores and no inter-core communication would occur.

For the evaluation, Round-Robin, and two Nearest-Neighbor run-time mapping heuristics are applied. As a reference, random mapping is used. The Round-Robin (RR) maps to the first idle core in a round-robin fashion, therefore generating a moderate degree of locality in the mapping decisions. The Nearest-Neighbor algorithms map to an idle core nearest to a given reference position [12]. The first Nearest-Neighbor maps nearest to the location of the parent task. The second algorithm, called Near-Data (ND), maps nearest to the focal point of the input data. The Random and Round-Robin mapping both have $\mathcal{O}(1)$ time complexity. For the Nearest-Neighbor mappings a linear algorithm having $\mathcal{O}(m)$ time complexity has been implemented. The DRACON-AC implementation uses a linear hardware search engine to accelerate the mapping decisions (see Sec. III-E).

Generally, task mapping can not be considered independently of task scheduling, i.e. the task selection can



(a) Gain G_{real} of priority scheduling using real-overhead RTM for DRACON-AC



(b) Ratio of DRACON-AC vs. SW-Symm using priority scheduling

FIGURE 12. Scheduling.

have considerable influence to the subsequent locality of the task mapping. For this evaluation, First Come First Serve (FCFS) scheduling has been selected for the given mapping benchmark. Due to the horizontal task dependencies, the FCFS scheduling generates a depth-first order preferring tasks inside a sequence before starting a new sequence. For the given evaluation, the benchmark consists of $p = 16$ sequences, each sequence having 8 tasks. The mean task size has been set to $l = 32000$. The mean communication volume between two tasks amounts to 8000 bit. The task sizes and communication volume have a random uniform distribution and a standard deviation of $\sigma_l = 800$ (task size) and $\sigma_v = 100$ (communication volume). As a parameter the communication volume between the tasks is varied.

Fig. 13 shows the gain of Round-Robin, Near-Parent and Near-Data mapping compared to random mapping for DRACON-AC. As shown, the Near-Data algorithm has a reasonable gain if the communication volume is increased. However, the overall speedup generally decreases if the communication volume is increased. Further, there is a break-even point between RR mapping and Nearest-Neighbor mapping: For small communication volume RR mapping performs better, for large communication volume Nearest-Neighbor mapping gives better results.

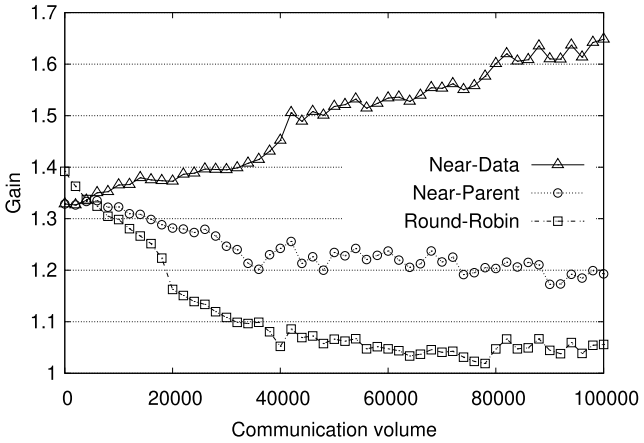


FIGURE 13. Task mapping: Gain vs. random mapping using DRACON-AC.

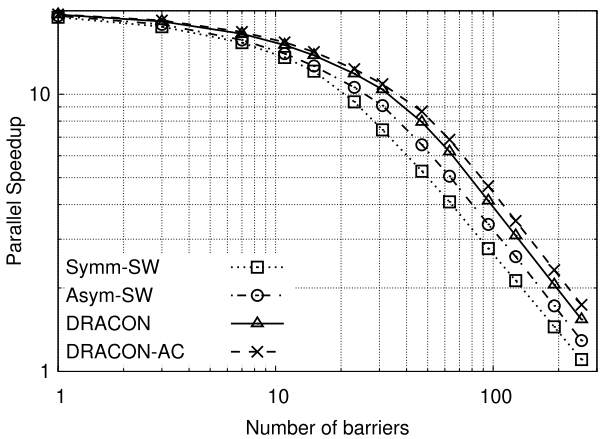
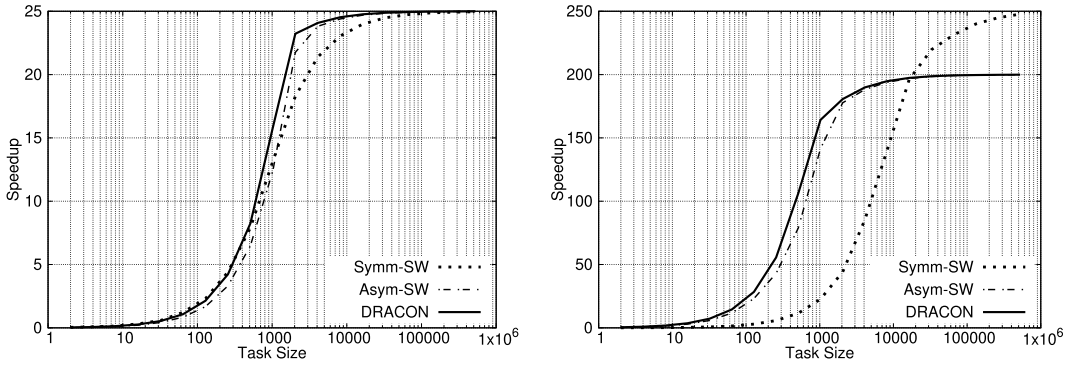


FIGURE 14. Synchronization: Varying barrier count.

3) SYNCHRONIZATION

To evaluate the impact of task synchronization to the application performance, a synthetic benchmark describing a barrier synchronization scenario was implemented (Fig. 11c). The number of barriers is varied while the total amount of work and the degree of parallelism is kept constant. In other words, the number of tasks increases proportionally to the number of barriers while the task granularity decreases. Fig. 14 shows the corresponding application speedup. The figures reveal the significant influence of the synchronization barriers to the application performance. DRACON-AC uses memory-mapped hardware FIFOs to accelerate task synchronization (see Sec. III-E). Due to the memory mapping overhead, there is further optimization potential for DRACON-AC by improving the interface to the hardware FIFOs (e.g. using a dedicated instruction set architecture).



(a) Single-Master (b) Multi-Master

FIGURE 15. Dispatching: Speedup versus task size.

4) DISPATCHING

Fast dispatching of user tasks is a crucial requirement in the many-core domain. To evaluate the dispatching capabilities of the proposed DRACON architecture a synthetic benchmark consisting of independent tasks is applied (Fig. 11a). For the run-time task management, a simple FCFS scheduling algorithm and a simple round-robin mapping is used. The task dispatching operates in a distributed environment having multiple RTM masters. The recursive application start-up mechanism (see Sec. III-C.3) is applied. The evaluation omits DRACON-AC as it does not contain additional implementation for task dispatching compared to DRACON.

In the experiment, the task size is varied while only one cluster of 32 cores is used. The number of tasks is set to $n = 100$. Fig. 15a shows the speed-up versus the task size using a single RTM master for SW-Asym and DRACON. The speedup is limited by the management overhead for small-sized user tasks. For larger user tasks, the speedup raises up to theoretical maximum $S = n \cdot l / \lceil n/m \rceil \cdot l = 100/\lceil 100/32 \rceil = 100/4 = 25$. All three implementations have comparable performance and a steep increase in application speedup if the task size grows around 1000.

Fig. 15b shows the achievable application speedup when applying multiple clusters of cores. The number of overall processing cores is a constant and set to 256 cores. For DRACON and the asymmetric software RTM the number of RTM masters is set to $k = 16$. The number of user tasks is increased to $n = 1000$. Notably, the SW-Asym and DRACON have a better performance for task sizes of $l \approx 1000$, however have a maximum speedup of $S = 200$ for larger user tasks. For large user tasks, the speedup of the symmetric software RTM gets close to the theoretical maximum of $m = 256$ due to the larger number of user cores.

5) RTM MASTER

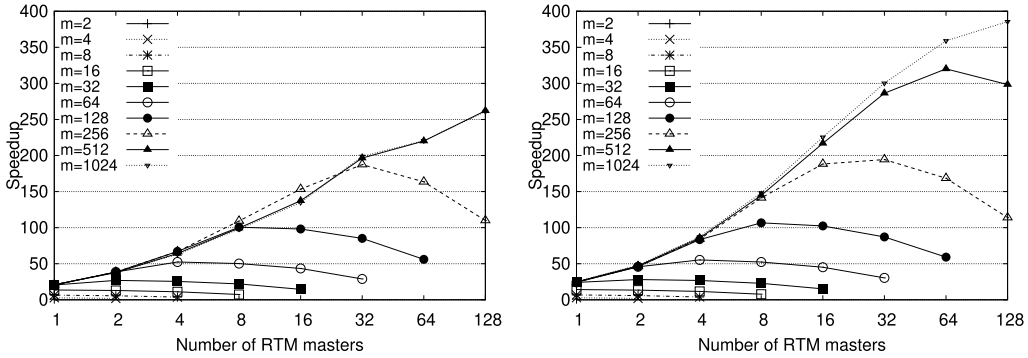
The number of RTM masters and their implementation can have significant impact to the overall system performance. Therefore, an analysis varying the number of RTM masters while keeping the number of cores constant is performed. A benchmark consisting of horizontal task dependencies (Fig. 11b) and a large number of concurrent sequences ($p = 2000$) is applied. Fig. 16a, and 16b show the results for SW-Asym, and DRACON. On the average, one RTM master per 16 cores appears to be optimal. The evaluation indicates the feasibility and usefulness of asymmetric RTM architectures. While increasing the number of cores, increasing the number of RTM masters maintains a reasonable application speedup. DRACON has a considerably better performance compared to SW-Asym. Therefore, dedicated interconnects potentially further improve the scalability of RTM implementations.

C. MCSL BENCHMARKS

The following section applies MCSL task graph benchmarks which are derived from real-world applications [37]. In particular the task graphs for a sparse matrix solver (Sparse), a robot control (Robot), Fpppp a quantum chemistry application, and H264 video decoder are used. The actual mapping of the tasks is computed by the RTM at run-time. Therefore, the evaluation does not consider the static task mappings which are proposed by MCSL, but only the task graph models. The evaluation compares between the RTM architectures SW-Symm, SW-Asym, DRACON and DRACON-AC. In addition to the regular H264 benchmark, a modified version, the H264p is applied. The regular H264 has a large degree of concurrency, however it also has a dominating sequential section resulting into a limited maximum speedup. To show the speedup capabilities of the highly concurrent H264 section, the modified version (H264p) is applied where the sequential bottleneck was removed. The characteristics of the benchmark applications (maximum speedup S_{max}, number of task `ncnt`, number of edges `ecnt`, accumulated computation time `twrk` and accumulated communication volume `cvol` in 32bit words) is shown in Tab. 3.

1) SINGLE-PROGRAM

At a first instance, only single program benchmarks are applied, i.e. only one instance of a MCSL benchmark is running at a time. As a metric, the parallel application



(a) SW-Asym (b) DRACON

FIGURE 16. Evaluation of optimal number of RTM masters.

TABLE 3. Benchmark characteristics [37].

Benchmark	S_{max}	ncnt	ecnt	twrk	cvol
Sparse	15.87	96	67	309760	13668
Robot	4.36	88	131	397280	6681
Fpppp	6.71	334	1145	570320	58395
H264	7.30	2311	3461	141203442	1107520
H264p	767.97	2311	3461	121934599	1107520

speedup S is applied while varying the number of cores and RTM masters. By default, priority-based Max-Bottom-Level-First scheduling [48] and Round-Robin (RR) mapping is applied.

Fig. 17 shows the result for the MCSL benchmarks. At m = 512 cores, DRACON-AC using ND mapping achieves an improvement of 15.21% vs. SW-Asym. Notably, the Near-Data (ND) mapping algorithm can only be exploited by the DRACON-AC architecture due to its high RTM overhead. However, RR mapping reveals comparable performance for the MCSL benchmarks while having lower complexity. For the MCSL Sparse and Fpppp benchmark DRACON-AC achieves best results using either RR or ND mapping. For H264p SW-Asym, DRACON and DRACON-AC using RR mapping achieve similar performance to DRACON-AC using ND mapping. Further, for H264p the SW-Symm RTM using either RR or ND mapping does not scale well beyond $m \geq 256$ cores.

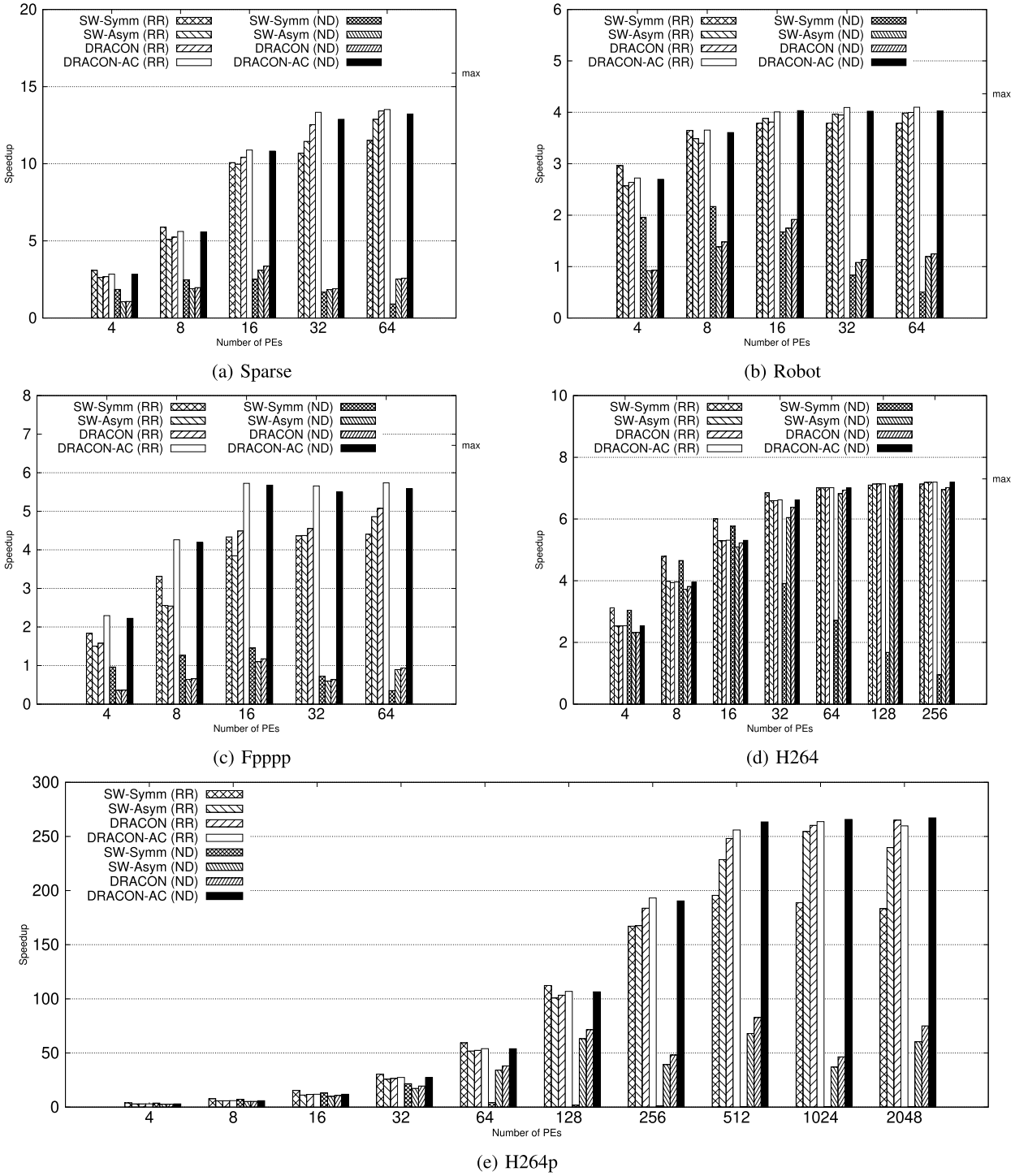
2) RANDOM INJECTION

The following analysis considers the periodic injection of the Sparse, Robot, Fpppp and H264 workloads. Each workload has a probability of 0.25 to be launched at the beginning of an injection interval. The length of an injection interval thereby has a Poisson distribution. Due to memory restrictions of Agamid, especially for large configurations ($m \geq 1024$), only a limited amount of injection intervals can be simulated. To get a more meaningful result, the output is averaged over multiple simulation runs (samples). Nevertheless, simulations considering a large system and a high injection frequency potentially have a lower accuracy. According to the preceding evaluations, priority-based scheduling and RR mapping is applied for all the RTM architectures.

Fig. 18 shows the normalized application speedup for an injection frequency of $(10^{-8}/ns)$ with respect to a varying number of cores. As additional reference models, SW-Asym-GI having a global common interconnect and MS-HW having a single but highly optimized RTM masters are considered in the this section. While SW-Asym may correspond to fos [53], SW-Asym-GI has a strong similarity to STHORM [39], MS-HW can be seen as a counterpart to Nexus++ [15]. As a zero reference measurement, Ref-Zero shows the application performance when using a centralized, zero overhead RTM. The results indicate a rising impact of RTM communication for a larger number of cores. The more cores, the smaller the performance difference between DRACON and DRACON-AC. DRACON and DRACON-AC reveal a much better scalability compared to the symmetric and asymmetric software references: 6.12% improvement for DRACON-AC vs. SW-Asym at $m = 256$ and 6.27% at $m = 1024$. While SW-Symm suffers from lock contention with a larger number of cores, SW-Asym and SW-Asym-GI suffer from contention at the common interconnect between user and RTM. Thereby, high priority RTM communication may block and slow down user communication.

D. GATE-LEVEL ANALYSIS

Tab. 4 provides values for the gate-level area of HMSTs, HSLVs and user processing cores. The values have been obtained using an industrial 65nm low-power process. Each HMST and each user processing core contains one dedicated RISC processor which is implemented as mLite/PLASMA CPU [45]. 32-Bit RX buffers for message communication are considered inside each HMST. Assuming a system of 256 processing cores including 16 HMSTs the RX buffers have a size of 16 entries which gives one RX entry per connected core at the local and the global management interconnect. To address the impact of on-chip memory 4kB of SRAM per user processing core and HMST are included. Eqn. 2 can be applied to estimate the area overhead of DRACON (where m is the overall number of processing



(a) Sparse

(b) Robot

(c) Fpppp

(d) H264

(e) H264p

FIGURE 17. MCSL benchmarks including task communication.

cores, and k is the number of hardware enhanced RTM masters and $A[x]$ is the area of design unit x). Considering 256 processing cores and 16 RTM masters, DRACON exhibits an area overhead of 3.01%.

$$\Omega_{area} = \frac{(m-k)\cdot(A[Core]+A[HSLV])+k\cdot A[HMST]}{m\cdot A[Core]} \quad (2)$$

V. CONCLUSION

The many-core regime raises novel paradigms for hardware and software designers: The impossibility of full transistor utilization (dark silicon), an increasing number of fine-grained user tasks, and 2D/3D network-on-chips connecting a large number of cores. Therefore, the requirements to architectures and algorithms for run-time task and power

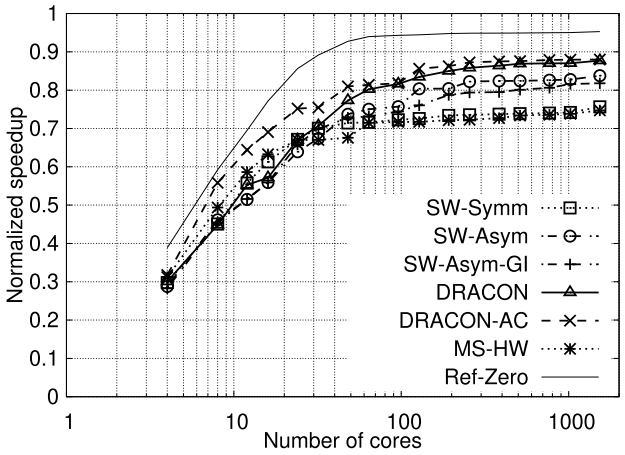


Figure 18: Random injection and varying number of cores

FIGURE 18. **Random injection and varying number of cores.**

TABLE 4. **Area for 65nm technology [μm^2].**

Unit	Total	Comb.	Non-comb.	f_{max}
HMST	81003.8	20072.8	60931.0	502.5 MHz
core	30397.3	17313.1	13084.2	
mem	50606.5	2759.7	47846.8	
Core	63367.9	14246.2	49121.7	529.1 MHz
core	27627.8	14241.9	13385.9	
mem	35740.1	4.3	35735.8	
HSLV	858.6	375.5	483.1	543.4 MHz

management also increase. Asymmetric software architectures (e.g. fos [53]) provide an advancement in scalability compared to symmetric architectures (e.g. Linux), however both suffer from software-induced management latencies.

This work contributes the DRACON architecture, a dedicated infrastructure for hardware enhanced run-time management. Purpose of the parallel RTM hardware is the elimination of user interferences and the significant mitigation of RTM overhead. The DRACON architecture facilitates dedicated hardware resources for RTM computation, dedicated interconnects and a low-latency interface between the user cores and the RTM. Additionally, DRACON allows the integration of distributed hardware accelerators for RTM algorithms.

As a use case for the DRACON architecture, dynamic task management is considered. The evaluation of DRACON and reference implementations proves the efficiency of using a dedicated infrastructure to enhance computation and communication for RTM in the many-core domain. Using DRACON achieves an improvement of up to 15.21% for single-program and 6.12% - 6.27% improvement for multi-program. For 256 processing cores including 16 RTM masters, an affordable area overhead of 3.01% is approximated. In contrast to previous works (e.g. Nexus++ [15], IsoNet [35], STHORM [39]), DRACON provides improved scalability, performance and task management functionality. A symmetric software reference revealed a scalability bottleneck due to global RTM lock contention. Also, symmetric architectures may suffer from a large, global view search space of the management algorithms. On the other hand, fully distributed approaches suffer from a higher communication overhead. Therefore, a clustered approach appears necessary to enable scalability for many-cores. However, reference implementation of a clustered and asymmetric software RTM revealed limited scalability due to contention between user and RTM communication at the interconnects. In comparison, DRACON improves scalability by using dedicated interconnects for RTM communication and distributed hardware accelerators for low-overhead and high-quality RTM management algorithms. Consequently, it appears worthwhile to add a dedicated infrastructure for many-core RTM. Subsequent research approaches evolve for adaptive implementations of hardware enhanced resource management.

REFERENCES

[1] J. Adomat, J. Furunas, L. Lindh, and J. Starner, "Real-time kernel in hardware RTU: A step towards deterministic and high-performance real-time systems," in *Proc. 8th Euromicro Workshop Real-Time Syst.*, Jun. 1996, pp. 164–168.
[2] H. Akkan, M. Lang, and L. Liebrock, "Understanding and isolating the noise in the Linux kernel," *Int. J. High Perform. Comput. Appl.*, vol. 27, no. 2, pp. 136–146, Feb. 2013.
[3] *AMBA Specification Rev 2.0*, ARM, Cambridge, U.K.,1999.
[4] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from Berkeley," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2006-183, 2006.
[5] A. Baumann, P. Barham, P. E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, A. Singhania, "The multikernel: A new os architecture for scalable multicore systems," in *Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Princ.*, Oct. 2009, pp. 29–44.
[6] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Proc. Design, Automat. Test Europe Conf. Exhib. (DATE)*, Mar. 2012, pp. 983–987.
[7] S. Borkar, "Thousand core Chips—A technology perspective," in *Proc. DAC*, Jun. 2007, pp. 746–749.
[8] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang, "Corey: An operating system for many cores," in *Proc. 8th USENIX Conf. Oper. Syst. Design Implement.*, Dec. 2008, pp. 43–57.
[9] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Morris, R. Morris, and N. Zeldovich, "An analysis of Linux scalability to many cores," in *Proc. OSDI*, Oct. 2010, pp. 1–16.
[10] W. Büter, C. Osewold, A. Ahmed, D. Gregorek, and A. Garcia-Ortiz, " Predictable photonic interconnects using an autonomous channel management and a TDMA-NoC," in *Proc. 10th Int. Symp. Reconfigurable Commun.-Centric Syst.–Chip (ReCoSoC)*, Jun./Jul. 2015, pp. 1–6.
[11] W. Büter, C. Osewold, D. Gregorek, and A. Garcia-Ortiz, "DCM: An IP for the autonomous control of optical and electrical reconfigurable NoCs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2014, pp. 1–4.
[12] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs," in *Proc. 18th IEEE/IFIP Int. Workshop Rapid Syst. Prototyping (RSP)*, May 2007, pp. 34–40.
[13] J. A. Colmenares, S. Bird, H. Cook, P. Pearce, D. Zhu, J. Shalf, S. Hofmeyr, K.Asanović, and J. Kubiatowicz, "Resource management in the tessellation manycore OS," in *Proc. HotPar*, vol. 10, Jun. 2010, pp. 1–6.
[14] T. Dallou, N. Engelhardt, A. Elhossini, and B. Juurlink, "Nexus#: A distributed hardware task manager for task-based programming models," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2015, pp. 1129–1138.
[15] T. Dallou and B. Juurlink, "Hardware-based task dependency resolution for the starss programming model," in *Proc. 41st Int. Conf. Parallel Process. Workshops (ICPPW)*, Sep. 2012, pp. 367–374.
[16] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz, "CPU DB: Recording microprocessor history," *Commun. ACM*, vol. 55, no. 4, pp. 55–63, 2012.

[17] B. D. de Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P. G. de Massas, F. Jacquet, S. Jones, N. M. Chaisemartin, F. Riss, and T. Strudel, "A clustered manycore processor architecture for embedded and accelerated applications," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2013, pp. 1–6.

[18] B. D. de Dinechin, P. G. de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, "A distributed run-time environment for the Kalray MPPA-256 integrated manycore processor," *Procedia Comput. Sci.*, vol. 18, pp. 1654–1663, Jan. 2013.

[19] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 365–376.

[20] Y. Etsion, F. Cabarcas, A. Rico, A. Ramirez, R. M. Badia, E. Ayguade, J. Labarta, and M. Valero, "Task superscalar: An out-of-order task pipeline," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2010, pp. 89–100.

[21] M. A. A. Faruque, R. Krist, and J. Henkel, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," in *Proc. 45th Annu. Design Automat. Conf. (DAC)*, Jun. 2008, pp. 760–765.

[22] D. Gregorek and A. Garcia-Ortiz, "The DRACON embedded many-core: Hardware-enhanced run-time management using a network of dedicated control nodes," in *Proc. Int. Symp. VLSI (ISVLSI)*, Jul. 2015, pp. 416–421.

[23] D. Gregorek and A. Garcia-Ortiz, "The Agamid design-space exploration framework task-accurate simulation of hardware-enhanced run-time management for many-core," *Des. Automat. Embedded Syst.*, vol. 22, no. 4, pp. 293–314, Dec. 2018.

[24] D. Gregorek, C. Osewold, and A. Garcia-Ortiz, "A scalable hardware implementation of a best-effort scheduler for multicore processors," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Sep. 2013, pp. 721–727.

[25] D. Gregorek, R. Schmidt, and A. Garcia-Ortiz, "Transaction level analysis for a clustered and hardware-enhanced task manager on homogeneous many-core systems," Feb. 2015, *arXiv:1502.02852*. [Online]. Available: https://arxiv.org/abs/1502.02852

[26] J. Heiβwolf, A. Zaib, A. Weichslgartner, R. König, T. Wild, J. Teich, A. Herkersdorf, and J. Becker, "Hardware-assisted decentralized resource management for networks on chip with QoS," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops PhD Forum*, May 2012, pp. 234–241.

[27] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," in *Proc. 52nd ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, Jun. 2015, pp. 1–6.

[28] A. Herkersdorf, A. Lankes, M. Meitinger, R. Ohlendorf, S. Wallentowitz, T. Wild, and J. Zeppenfeld, "Hardware support for efficient resource utilization in manycore processor systems," in *Multiprocessor System–Chip*. New York, NY, USA: Springer, 2011, pp. 57–87.

[29] S. J. Hollis and S. Kerrison, "Overview of swallow—A scalable 480-core system for investigating the performance and energy efficiency of many-core applications and operating systems," Apr. 2015, *arXiv:1504.06357*. [Online]. Available: https://arxiv.org/abs/1504.06357

[30] S. J. Hollis, E. Ma, and R. Marculescu, "nOS: A nano-sized distributed operating system for many-core embedded systems," in *Proc. IEEE 34th Int. Conf. Comput. Design (ICCD)*, Oct. 2016, pp. 177–184. doi: 10.1109/ICCD.2016.7753278.

[31] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart, "A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[32] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, "DistRM: Distributed resource management for on-chip many-core systems," in *Proc. 7th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2011, pp. 119–128.

[33] O. Krieger, M. Auslander, B. Rosenburg, R. W. Wisniewski, J. Xenidis, D. D. Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig, "K42: Building a complete operating system," in *Proc. 1st ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, Apr. 2006, pp. 133–145.

[34] S. Kumar, C. J. Hughes, and A. Nguyen, "Carbon: Architectural support for fine-grained parallelism on chip multiprocessors," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 162–173, May 2007.

[35] J. Lee, C. Nicopoulos, H. G. Lee, S. Panth, S. K. Lim, and J. Kim, "IsoNet: Hardware-based job queue management for many-core architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 1080–1093, Jun. 2013.

[36] Y. Lhuillier, M. Ojail, A. Guerre, J.-M. Philippe, K. B. Chehida, F. Thabet, C. Andriamisaina, C. Jaber, and R. David, "HARS: A hardware-assisted runtime software for embedded many-core architectures," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 13, no. 3, Mar. 2014, Art. no. 102.

[37] W. Liu, J. Xu, X. Wu, Y. Ye, X. Wang, W. Zhang, M. Nikdast, and Z. Wang, "A NoC traffic suite based on real applications," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2011, pp. 66–71.

[38] T. Miyamori, H. Xu, T. Kodaka, H. Usui, T. Sano, and J. Tanabe, "Development of low power many-core soc for multimedia applications," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2013, pp. 773–777.

[39] J. Mottin, M. Cartron, and G. Urlini, "The STHORM platform," in *Smart Multicore Embedded System*. New York, NY, USA: Springer, 2014, pp. 35–43.

[40] S. Muir and J. Smith, "AsyMOS-an asymmetric multiprocessor operating system," in *Proc. IEEE Open Archit. Netw. Program.*, Apr. 1998, pp. 25–34.

[41] D. Nellans, R. Balasubramonian, and E. Brunvand, "A case for increased operating system support in chip multi-processors," in *Proc. 2nd IBM Watson*, Sep. 2005, pp. 1–10.

[42] V. Nollet, D. Verkest, and H. Corporaal, "A safari through the MPSoC run-time management jungle," *J. Signal Process. Syst.*, vol. 60, no. 2, pp. 251–268, 2010.

[43] S. Park, D.-S. Hong, and S. I. Chae, "A hardware operating system kernel for multi-processor systems," *IEICE Electron. Express*, vol. 5, no. 9, pp. 296–302, May 2008.

[44] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, D. Lyonnard, O. Benny, B. Lavigueur, D. Lo, G. Beltrame, V. Gagne, G. Nicolescu, "Parallel programming models for a multiprocessor soc platform applied to networking and multimedia," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 7, pp. 667–680, Jul. 2006.

[45] S. Rhoads. *Plasma-Most MIPS I (TM) Opcodes: Overview. Internet*. Accessed: May 2, 2012, [Online]. Available: http://opencores.org/project,plasmas

[46] M. J. Rutten, J. T. J. van Eijndhoven, E. G. T. Jaspers, P. van der Wolf, O. P. Gangwal, A. Timmer, and E.-J. D. Pol, "A heterogeneous multiprocessor architecture for flexible media processing," *IEEE Design Test Comput.*, vol. 19, no. 4, pp. 39–50, Jul./Aug. 2002.

[47] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. 50th Annu. Design Automat. Conf. (DAC)*, May 2013, pp. 1–10.

[48] O. Sinnen, *Task Scheduling for Parallel Systems*, vol. 60. Hoboken, NJ, USA: Wiley, 2007.

[49] G. H. Sockut, "Firmware/hardware support for operating systems: Principles and selected history," *ACM SIGMICRO Newslett.*, vol. 6, no. 4, pp. 17–26, Dec. 1975.

[50] M. B. Taylor, "Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse," in *Proc. 49th Annu. Design Automat. Conf.*, Jun. 2012, pp. 1131–1136.

[51] F. Thabet, Y. Lhuillier, C. Andriamisaina, J.-M. Philippe, and R. David, "An efficient and flexible hardware support for accelerating synchronization operations on the sthorm many-core architecture," in *Proc. Conf. Design, Automat. Test Eur.*, Mar. 2013, pp. 531–534. Autom

[52] A. Weichslgartner, J. Heisswolf, A. Zaib, T. Wild, A. Herkersdorf, J. Becker, and J. Teich, "Position paper: Towards hardware-assisted decentralized mapping of applications for heterogeneous NoC architectures," in *Proc. 28th Int. Conf. Archit. Comput. Syst. (ARCS)*, Mar. 2015, pp. 1–4.

[53] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): The case for a scalable operating system for multicores," *ACM SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 76–85, Apr. 2009.

DANIEL GREGOREK received the M.Sc. degree in computer engineering from the University of Hanover, Germany, in 2009, and the Ph.D. degree from the Institute of Electrodynamics and Microelectronics, University of Bremen, Germany, in 2018, where he currently holds a Postdoctoral position. His research interests include run-time management for many-core processors and system-on-chip design for mobile communications.

JOCHEN RUST received the diploma (Dipl.Ing.) degree in electrical engineering and computer science from the University of Hanover, in 2007, and the Ph.D. degree from the Institute of Electrodynamics and Microelectronics, University of Bremen, Germany, 2014. Since 2008, he has been with the Institute of Electrodynamics and Microelectronics. His research interests include high-performance computing, approximate computing, and hardware implementations of wireless industrial communication systems. In 2007, he received an Innovation Award from IBM for his contributions to tree-grammar-based netlist verification.

ALBERTO GARCIA-ORTIZ received the Diploma degree in telecommunication systems from the Polytechnic University of Valencia, Spain, in 1998, and the Ph.D. degree *(summa cum laude)* from the Institute of Microelectronic Systems, Darmstadt University of Technology, Germany, in 2003. He was with Newlogic, Austria. From 2003 to 2005, he was a Senior Hardware Design Engineer with IBM Deutschland Development and Research, Böblingen. He then joined the start-up AnaFocus, Spain, where he was responsible for the design and integration of AnaFocus' next-generation vision system-on-chip. He is currently a Full Professor with the Chair of Integrated Digital Systems, University of Bremen. In 2003, he received the Outstanding Dissertation Award from the European Design and Automation Association. He serves as an Editor for JOLPE. He is also a Reviewer of several conferences, journals, and European projects.