Partitioning and Task Transfer on NoC-based Many-Core Processors in the Avionics Domain

Robert Hilbrich and J. Reinier van Kampenhout
Fraunhofer FIRST, Berlin
{robert.hilbrich | j.r.van.kampenhout}@first.fraunhofer.de

***Abstract*—Networks-on-Chip (NoC) based many-core processors can not only increase system performance but also allow the integration of multiple functions on a single hardware platform. To consolidate functionality on many-core systems in safety-critical domains software partitioning is required to avoid the propagation of faults due to the use of shared resources.**

In this paper we propose extensions to well established single-core partitioning mechanisms to take novel architectural characteristics of many-core processors into account. In addition to fixed partitioning, we present *flexible* partitioning as an approach to improve resource utilization and fault tolerance using dynamic reconfiguration. Flexible partitioning requires task migration between cores via a shared resource - the NoC - which may endanger the required predictability. Therefore we empirically analyze a variety of task transfer mechanisms on a Tilera TILE*Pro*64™ many-core processor regarding their potential for deterministic reconfiguration during run-time.

***Keywords*—Avionics, manycore, partitions, task migration**

I. Introduction

The ongoing trend towards processors with many computational cores (more than 32) and its effect on software development has received much attention lately. Increased performance and a reduced power consumption are regarded as the main drivers for the adoption of many-core processors, which can be achieved by exploiting concurrency and scalability in software.

The interest in many-core technology of the aviation electronics (avionics) industry is not only driven by performance reasons. The increasing complexity and unpredictability of general purpose processors leads to a significant interest in chips with simple and predictable cores, which would reduce the effort required for timing analysis. When many of such execution units are integrated on one chip it becomes possible to consolidate the software of multiple single-cores onto a few many-core devices. This cuts production and operation costs because it reduces the space, weight and power (SWaP) requirements.

Many-core processors also introduce new challenges, especially in software engineering. Research challenges encompass not only the efficient parallelization of legacy software, but also efficient deployment schemata and reliability concerns. In our research we focus on the applicability of many-core processors in safety critical embedded systems, such as avionics equipment.

Integrating multiple functions on a single-core processor is a common design practice in avionics systems. This might however lead to *non-transparent fault propagation*, which significantly reduces the reliability of the device. Therefore robust *partitioning* in time and space is used for single-core processors to avoid the propagation of faults. Partitioning guarantees that a fault in one software partition cannot affect an application in another. The question arises if traditional temporal and spatial partitioning can be used for many-core processors as well in order to achieve a similar level of software isolation.

Our contribution is three-fold. First, we present an overview of the requirements for the development and deployment of safety-critical software in the avionics domain. Second, we propose to extend the single-core partitioning concept to facilitate the consolidation of functions on NoC-based many-core processors. Third, we analyze the benefits of deterministic reconfiguration and evaluate different transfer mechanisms on a many-core processor empirically.

II. Related Work

In [1] the differences between federated avionic architectures and Integrated Modular Avionics (IMA) are presented. The authors particularly depict increased resource utilization and SWaP savings as the benefits of transitioning to IMA.

The ongoing trend towards many-core architectures with "*1000 cores*" is described and motivated with several challenges arising from increased technology scaling in [2].

In [3] the componentization and partitioning of tasks is proposed to facilitate the transition of embedded real-time systems to multi-core processors. We pursue a similar approach but use a different architecture, i.e. a NoC-based many-core processor, and we do not focus on the migration of existing software.

The Networks-on-Chip design presented in [4] shows that services such as uncorrupted and ordered data delivery, guaranteed throughput and bounded latency can be delivered with contention-free routing. This motivates the use of NoC-based many-core processors in safety critical domains, where deterministic transfer times are required.

Support for task migration in many-core processors with a user-managed migration scheme based on code checkpointing is proposed in [5]. It is shown that there is a break-even point after which the migration cost is compensated as a result of load balancing.

In [6] the impact of task migration in multi-cores is analyzed. It is demonstrated that migration helps meeting

soft real-time deadlines without performance degradation or increased energy consumption.

III. Software Partitioning in Avionics Systems

Among the dominating trends are increasing functional requirements, the demand for more computer-based systems and the need for a shorter time to market. At the same time, it is a safety-critical domain where human lives are at stake.

In the past avionics systems were based on heterogeneous *federated architectures*. A distinctive feature of these architectures is that each function has its own independent computer system - "*one function - one computer*". A major advantage of this approach is that faults are contained *by design* due the use of dedicated components. Federated architectures however exhibit significant drawbacks. Firstly, many dedicated resources are required which raises costs for procurement, cooling, and maintenance and increases the SWaP requirements. Secondly, aircraft functionality is artificially separated into independent components without global control or synchronization, thus rendering the implementation of complex functionality challenging and costly.

Driven by the economic considerations described above, the avionics domain is slowly transitioning from federated avionics architectures to an Integrated Modular Avionics (IMA) architecture [1]. IMA describes the concept of computing modules with standardized components and interfaces to hardware and software. Thus IMA constitutes a logically centralized and shared computing platform, which is physically distributed on the aircraft to meet redundancy requirements. This transition is motivated by the expected benefits of hosting a variety of avionics functions on a single computing platform - "*multi-function integration*".

Despite these advantages the use of common components in IMA architectures significantly increases the probability of common cause errors affecting multiple functions. Therefore federated architectures are still considered as the benchmark for fault containment [7].

To consolidate multiple avionics functions on a single-core processor IMA requires the use of *software partitioning*. This is a concept to achieve fault containment in software, independent of the underlying hardware platform. Faults may propagate from one application to another through shared resources, such as a processor, memory, communication channels or I/O devices. Partitioning isolates faults by means of access control and usage quota enforcement for resources in software.

In avionics safety standards this is referred to as *partitioning in time and space* (cf. RTCA DO-178B, page 9). *Spatial* partitioning ensures that an application in one partition is unable to change private data of another. It also ensures that private devices of a partition, i.e. actuators, cannot be used by an application from another partition. *Temporal* partitioning on the other hand guarantees that the timing characteristics of an application, such as a worst case execution time, are not affected by the execution of an application in another partition. Time and Space Partitioning result not only in increased reliability, but also improves the predictability because unpredictable interferences with non-deterministic delays are reduced.

Implementing software partitioning and integrating multiple aircraft functions is not trivial, as there are different gradations in the criticality level of avionics software. Each criticality level implies specific safety precautions and imposes restrictions on the scheduling of partitions and processes on the underlying computing platform.

Furthermore avionics applications differ in their execution model. There are *synchronous applications* which occur periodically and must be statically scheduled based on a fixed scheduling scheme that is continuously repeated. Such applications are usually assigned a high criticality level, which means their behavior must be fully deterministic.

Asynchronous applications on the other hand are event or data driven. Fixed scheduling of such applications cannot account for the dynamic behavior. Because fixed scheduling is still the standard for partitions, conservative estimations of the required resources are used for scheduling. This results in inefficient use of the available hardware resources.

The consolidation of both types of applications with different criticality levels on the same platform using partitioning is therefore a challenging task.

Current IMA processor boards usually contain a single processor with one execution unit. With many-core processors becoming available, the potential for consolidation of applications is even more promising.

IV. Many-Core Processors & On-Chip Communication Architectures

Multi- and many-core processors refer to the integration of multiple computational units ("*cores*") on a single die. Thus the throughput of a processor is increased, in contrast to methods that speed up the peak performance.

Technology scaling leads to an increasing gap between interconnection delays and gate delays [8]. Therefore the communication capacity becomes a performance bottleneck in VLSI design. Wire delays and timing uncertainties pose problems as the transistor density and clock frequency increase.

The "globally asynchronous, locally synchronous" paradigm offers a solution to this problem by separating local clock domains [9]. A global interconnect is required for inter-domain communication. Bus-based structures are not suitable for such on-chip interconnects because they lack scalability for an increasing amount of cores, instead designers turn to the use of packet-based Networks-on-Chip (NoC) [10].

Such networks solve the clock skew problem and offer flexibility as well as high bandwidth [11]. The cost of the interconnect in terms of silicon and power consumption however rises significantly. This leads to *communication-centric* designs as opposed to traditional *computation-centric* designs (see [12]).

Thus the on-chip interconnect becomes the dominating resource in many-core processors. It is not only used for

inter-core communication but also offers access to off-chip components such as the main memory. Every data item that a core needs must be transferred over the interconnect. This causes the execution time of software to depend heavily on those transfer times. The time required to access a resource furthermore depends on the location of the core because packet-switching uses multi-hop routing. This means such many-core processors are *Non-Uniform Memory Access* (NUMA) architectures.

The challenge of applying time and space partitioning to many-cores is to address the on-chip interconnect. Therefore we will have a closer look at *Networks-on-Chip*, and focus on deterministic communication in particular.

A Network-On-Chip usually consists of network interfaces that connect the cores, a number of routers, and a network of wires interconnecting the routers. Routers and wires are components that are shared by all cores and can thus be subject to contention and congestion. These effects render the performance of the network unpredictable , which may result in significant variations in the communication time of tasks.

The degree to which a NoC can satisfy the communication needs of the cores is named the *Quality of Service (QoS)*. Guarantees on QoS can be provided when certain properties of the traffic such as bandwidth and latency are within bounds. To achieve this congestion must be controlled or eliminated alltogether. One approach to guaranteed QoS is to implement *circuit switching* which is free of contention. Much of the benefits that NoCs bring are however cancelled by such a conservative approach because of overhead and the fact that wires are not shared anymore. These disadvantages can be overcome with Time Division Multiplexing (TDM), which has been proven to be able to provide guarantees on QoS [4].

V. Partitioning on Many-Core Processors

In order to further pursue the integration of functionality from separate devices onto NoC-based many-core processors, we have to address the challenge of using shared resources in applications that are executed concurrently. Therefore it is necessary to extend the definition of *time and space partitioning* to reflect the novel hardware capabilities of NoC-based many-core processors as described in section IV.

In avionics, a *partition* describes an isolated group of tasks with a certain criticality level. We suggest to extend time and space partitioning properties by incorporating the assignment ("*mapping*") of partitions and tasks to cores and the reservation of dedicated communication channels on the NoC. Mapping partitions and their tasks to cores can be easily implemented statically in configuration tools or dynamically as part of an operating system scheduler.

The reservation of communication capacity on the other hand is challenging and requires considerable effort for the following reasons. Firstly, the communication profile of each task must be analyzed. Secondly, either the hardware must feature reservation mechanisms such as circuit switching or such a scheme must be implemented in software. In relation to this the traffic must be isolated but robust isolation from faulty partitions can only be obtained with hardware mechanisms such as firewalls.

Partition Types

If both initial requirements for time and space partitioning on many-core processors can be satisfied, the consolidation of *synchronous* and *asynchronous* avionic applications on the same platform can be achieved. Depending on the use case and its safety requirements, we propose three different types of partitions: *fixed*, *mode-based* and *flexible* partitions. They differ in criticality and their ability to adapt to changing resource requirements at run-time.

Fixed Partitions: Those only contain synchronous tasks and are statically scheduled and mapped. Fixed partitioning offers a high degree of predictability, because all resources needed are determined and reserved at design-time. Dedicated cores and time slots in the communications channels allow the integration of highly critical tasks on a NoC-based many-core platform.

Mode-based Partitions: Partitions of this type are similar to fixed partitions, but by acknowledging the fact that an aircraft may reside in different (flight-) *modes* with varying resource requirements, mode-based partitioning improves resource utilization. On the ground, resources allocated to flight control systems may be different compared to landing or take-off phases. For each mode mappings and static schedules are determined and fixed at design-time which allows for a high degree of predictability. However, with mode switching at run-time, resource utilization can be optimized by adapting mapping and scheduling schemes to run-time events.

Flexible Partitions: Asynchronous tasks with lower criticality levels may benefit from being contained in a *flexible* partition. As a result of the functional homogeneity of the computational cores and excess NoC capacity, a high degree of flexibility can be attained by dynamically reconfiguring less critical partitions without affecting the highly critical fixed partitions. Aspects of a flexible partition which could be dynamically reconfigured are the *size* and the *shape* of a partition. The size of the partition relates to the number of cores, the communication links and communication bandwidth assigned to it. The shape describes its mapping onto the NoC. Resource requirements of applications in flexible partitions vary because of their asynchronous behaviour. Temporal variations occur because new data arrives and needs to be processed with low latency.

A major benefit is based on the fact that resources may be temporarily unallocated because applications in fixed partitions finish ahead of their deadline or asynchronous partitions are idle, waiting for data. A flexible partition can *borrow* these resources to finish earlier, thus increasing the overall resource utilization. Furthermore, it may also be beneficial to *transform* a partition by relocating some of its tasks. The goal of transforming a partition is to decrease the latency and distance to a resource it needs to interact

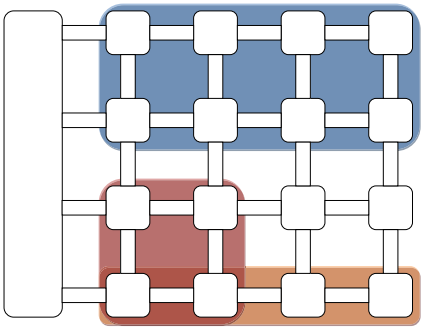

Fig. 1. A hypothetical many-core processor with 16 cores. It contains a fixed partition (blue) comprising of eight cores and a flexible partition comprising of four cores which transforms its shape dynamically (orange to red).

with. This reduces overall network traffic and congestion in a NoC. To illustrate this Figure 1 depicts a situation in which a flexible partition (orange) is transformed (red), while the resource allocation for the fixed partition (blue) remains the same. An important reason for transforming a partition is to increase the communication bandwidth and reduce the latency for access to off-chip resources such as memory or I/O located on the left side of Figure 1.

In fault-tolerant systems another benefit enabled by dynamic reconfiguration becomes apparent. Due to radiation, transient or permanent faults may occur in any part of a many-core processor which affects the functionality of cores or links. Upon detection, recovery measures must be taken to ensure correct operation of the system. In case of a transient fault, a corrupted software partition may be recovered by duplicating it and initiating a restart. Permanent faults can be dealt with by reconfiguring a flexible partition to avoid the use of faulty components.

However, dynamic reconfiguration of flexible partitions should not be used without further consideration at runtime. Prior to triggering a reconfiguration, the additional overhead induced by the process of reconfiguration itself needs to be analyzed in order to determine whether the expected benefits of reconfiguration outweigh its costs.

In our research we study the *coexistence* of fixed and flexible partitions on one many-core processor. We believe that the approach of using flexible partitions in addition to fixed and mode-based partitions can optimize the overall resource utilization on a shared computing platform.

In our experiments we focus on *task migration* which is required for both mode-based partitions and the transformation and relocation of flexible partitions. The use of *deterministic* task migration mechanisms makes it possible to assess the feasibility of dynamic reconfiguration of partitions. It has become clear that mode-based and flexible partitioning can be used to increase the resource utilization of asynchronous avionics software on many-core processors.

VI. Task Migration on Many-Core Processors

Task migration describes the process of halting a task on a source node, transferring the code, data and state to a destination node, followed by restoring and restarting the task. In many-core processors tasks must be migrated between cores via the NoC which is shared with the other cores.

There are several aspects in which this scenario differs from classic inter-processor migration. Firstly, the size of the local memory is limited and the operating system functionality on each core are limited. Secondly the actual transfer of the task must now be performed via the NoC. In real-time systems task migration is only feasible when an upper bound can be determined on the time required for every step of the migration process. Therefore all operations must be deterministic, which means the used transfer method must have guaranteed QoS with respect to latency, bandwidth and jitter. Because we see this as the major challenge in the implementation of mode-based and dynamic partitioning, we focus on deterministic task transfer over a NoC in our experiments.

VII. Task Transfer Experiments

In our experiments we compare different task transfer methods in order to determine which of those are deterministic. We implemented a basic task migration mechanism which transfers the code and data of a task over the interconnect.

Our test platform consists of a Tilera TILE*Pro*64™which is a homogeneous many-core processor featuring 64 *tiles*. Each tile contains a core running at 700 MHz, a two level cache and a router containing a fully connected crossbar switch. There are separate L1 instruction and data caches on each tile as well as an L2 cache of which the aggregrate can be used as a shared L3 cache.

The tiles are arranged in a 2-dimensional mesh structure and are interconnected with an iMesh NoC [13]. This NoC consists of six separate mesh networks and connects the tiles to each other, to the off-chip memory and to the I/O interfaces.

Data can be transferred either via shared memory or by directly sending packets over the NoC. The shared memory architecture is based on the shared coherent cache which uses three of the on-chip networks. End-to-end flow-control based on conservative buffer preallocation guarantees that the memory architecture and the networks that it uses are deadlock free. We use the shared cache system in three of the transfer methods.

Two of the networks can be accessed directly which enables “message passing” style communication. The User Dynamic Network (UDN) has dimension ordered (xy) routing while in the Static Network (STN) all switches must be programmed. The latter essentially implements circuit switching because connections must be set up and torn down, and contention is eliminated once a connection is established.

For our experiments we considered five different transfer methods on our test platform. The first three rely on the cache subsystem while the other two address the NoC directly. In the first method we use a cache-pull strategy in which data transfers are handled automatically by the cache

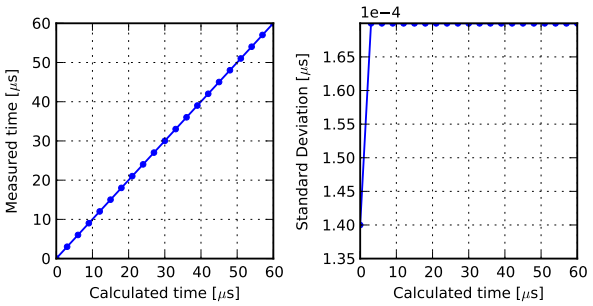

Fig. 2. Results of the timer test. On the left the measured time is compared with the calculated time, on the right the standard deviation is given.

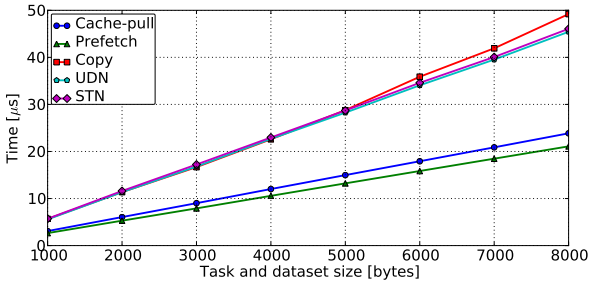

Fig. 3. The migration time plus one execution for different transfer methods, with varying task and dataset size

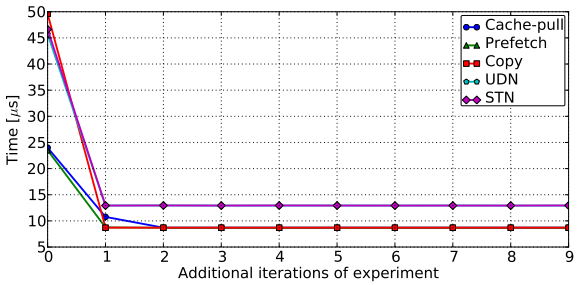

Fig. 4. The migration time plus execution time for multiple iterations of the experiment.

hardware. The obvious disadvantage is that a task might not access all of its data and code during the first execution on the destination core, thus spreading the migration over an unpredictable number of executions.

In the second transfer method prefetching is used to transfer the task to the destination core prior to the first execution, similar to [14]. Thus all data transfers appear in one block. The third method is very similar, but now data is explicitly copied instead of using prefetch instructions. This requires the destination core to store data in the local cache, which makes this method significantly slower than prefetching.

In the fourth method the source core packetizes the task and sends it over the UDN. Upon arrival the destination core stores the data into its local cache. The fifth method is similar except that the STN is used, this requires the additional setup of a route between two communicating partners.

In our test framework the transfer method and the task size can be adjusted. Furthermore high-precision timing measurements are carried out on dedicated cores. The processor is completely reserved for each experiment so the measurements are isolated. Tasks are designed to have a constant execution time. This is achieved by only using `NOP` and copy instructions so that the execution time of a task can be calculated very precisely. The size of the code and data is varied in steps of 1000 bytes.

The timer we use is based on a cycle-accurate counter that is available in each core. To test the precision of the timer we executed a number of `NOP` instructions in a loop. The execution time of this loop was calculated by looking at the assembly code and compared to the measured time. Each experiment was repeated 10.000 times after which the mean value and standard deviation were determined, the results are depicted in Figure 2.

The largest absolute difference between any two measurements is 17 ns, while the standard deviation is $< 0,2$ ns. We consider this adequate for the experiments presented in this report.

Results

We evaluated the migration time of the five different transfer methods in a series of experiments. In each experiment one migration is performed, followed by an execution of the migrated function and accessing all the elements in the migrated dataset. All tests are repeated 10.000 times, the mean value of the series is calculated to produce the graph. The maximum absolute deviation from the mean of all transfer methods is under 0,4 μs, which shows that a tight upper bound on the transfer time can be determined.

The results are depicted in Figure 3. We see that that all methods scale linearly with an increasing code and dataset size, and that there are no unexpected timing artifacts. Furthermore use of the cache hardware is clearly much faster than sending packets. The explanation for this is that the cache stores data automatically without intervention from the core, as opposed to the other methods. To compare the UDN and STN transfer methods with the memory system it is therefore more fair to look at the explicit copy method because it includes these copy instructions. These three curves are very similar. The time required to program the routers in the STN is not included in the measurements.

In the second series of experiments we investigate the effect of a migration on the execution time. The basic setup is the same as before but now a migration is executed only once, after which the task is repeatedly executed from the local cache on the destination core. The first measurement therefore consists of the migration time and one subsequent execution.

Figure 4 shows a migration and ten subsequent executions of the task on the destination core, the task and dataset size are 8000 bytes. We already learned that the migration time is much shorter when the cache subsystem is used. With the cache pull method however, we see

that the migration impacts the first *two* executions on the destination core due to unpredictable behavior of the cache hardware.

Analysis

Our experiments show that several transfer methods on our test platform can potentially be used for deterministic task migration. It became clear that the cache pull method is not suitable for use in real-time systems. The cache subsystem can however be manipulated with prefetch instructions, which avoids the unpredictable behavior and is by far the fastest. The memory networks are however not accessible by the programmer, which makes reservation and isolation of memory accesses and thus partitioning very difficult.

Use of the UDN is significantly slower because of the required store instructions, but the hardware is directly accessible so partitioning can be achieved by reserving resources in software. Thus "virtual" circuit switching can be implemented if the cores are synchronized, additional isolation of the circuits however still requires hardware support. Our test platform features the "hardwall" capability, which allows robust isolation of traffic by blocking all unauthorized traffic in the UDN and STN.

The latter implements circuit switching in hardware and is therefore naturally suited for partitioning and deterministic task transfer. A disadvantage is that switches must be reprogrammed when the circuits are changed, which requires communication via one of the other networks.

VIII. Conclusions

In this work we described the trend of consolidating software components on shared hardware platforms in the avionics domain by means of temporal and spatial partitioning. We propose to extend the concept of partitioning in order to exploit the benefits of current and future many-core processors. To deploy partitioned software on a many-core the on-chip interconnect must be reserved in time and space to offer the guarantees on QoS that are required in real-time systems. Transfer methods such as circuit switching are suited for this. Furthermore robust isolation between connections must be guaranteed, which is only possible with hardware support. We propose three approaches to software partitioning that each differ in the mapping of tasks and traffic onto hardware. Flexible mapping offers the potential to optimize hardware usage but requires that the mapping is changed during runtime. For this deterministic task migration is essential. We consider the dependence on communication the main new challenge in many-core computing and therefore focus our experiments on transfer methods. Our results show that deterministic task transfer is feasible, that is, the transfer of a task within a tight time bound. The use of the STN is a natural candidate for resource reservation and isolation.

Acknowledgements

This work is carried out as part of the VirtuOS project. The VirtuOS project is financed by TSB Technologiestiftung Berlin – Zukunftsfonds Berlin Co-financed by the European Union – European fund for regional development.

References

[1] C. Watkins and R. Walter, "Transitioning from federated avionics architectures to Integrated Modular Avionics," in *Digital Avionics Systems Conference, 2007. DASC '07. IEEE/AIAA 26th*, Oct. 2007, pp. 2.A.1–1–2.A.1–10.

[2] S. Borkar, "Thousand Core Chips - A Technology Perspective," in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, jun. 2007, pp. 746 –749.

[3] F. Nemati, J. Kraft, and T. Nolte, "Towards Migrating Legacy Real-Time Systems to Multi-Core Platforms," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, 2008, pp. 717–720. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4638477

[4] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 414–421, Mai 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1511973

[5] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali, "Supporting Task Migration in Multi-Processor Systems-on-Chip: A Feasibility Study," in *Proceedings of the Design Automation & Test in Europe Conference*. IEEE, 2006, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1656838

[6] E. W. Briao, D. Barcelos, F. Wronski, and F. R. Wagner, "Impact of task migration in NoC-based MPSoCs for soft real-time applications," in *2007 IFIP International Conference on Very Large Scale Integration*. IEEE, 2007, pp. 296–299. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4402516

[7] J. Rushby, "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," Langley, pp. 1–75, 1999. [Online]. Available: http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail$\backslash$\&id=oai:LTRS:NASA-99-cr209347

[8] ITRS, "International Technology Roadmap for Semiconductors: Interconnect," 2005.

[9] G. de Micheli and L. Benini, "Networks on Chip: A New Paradigm for Systems on Chip Design," in *DATE '02: Proceedings of the conference on Design, automation and test in Europe*. Washington, DC, USA: IEEE Computer Society, 2002, p. 418.

[10] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings of the 38th conference on Design automation - DAC '01*. New York, New York, USA: ACM Press, 2001, pp. 684–689. [Online]. Available: http://portal.acm.org/citation.cfm?doid=378239.379048

[11] J. Henkel, W. Wolf, and S. Chakradhar, "On-chip networks: a scalable, communication-centric embedded system design paradigm," in *17th International Conference on VLSI Design. Proceedings.* IEEE Comput. Soc, 2004, pp. 845–851. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1261037

[12] R. Hilbrich and R. van Kampenhout, "Dynamic reconfiguration in NoC-based MPSoCs in the avionics domain," in *IWMSE '10: Proceedings of the 3rd International Workshop on Multicore Software Engineering*. New York, NY, USA: ACM, 2010, pp. 56–57.

[13] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-c. Miao, J. F. Brown III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4378780

[14] A. Sarkar, F. Mueller, H. Ramaprasad, and S. Mohan, "Push-Assisted Migration of Real-Time Tasks in Multi-Core Processors," in *Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems - LCTES '09*. New York, New York, USA: ACM Press, 2009, p. 80. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1542452.1542464