Enhanced Reliability in Tiled Manycore Architectures through Transparent Task Relocation

Holm Rauchfuss, Thomas Wild, Andreas Herkersdorf

Technische Universität München
Lehrstuhl für Integrierte Systeme
Arcisstrasse 21
80290 München, Germany
holm.rauchfuss@tum.de
thomas.wild@tum.de
herkersdorf@tum.de

Abstract: Manycore platforms with tens and even up to hundreds of processing cores per chip are becoming a commercial reality and are subject of intensified research. This concept paper describes work in progress on the applicability of HW supported communication and processing virtualization on regular structured, tiled manycore architectures for the benefit of improved fault tolerance against transient and permanent perturbations. Temporarily unused, naturally redundant tiles are dynamically occupied during run time via transparent task relocation. This means, the execution of a task can pro-actively and transparently for the application be switched by distributed system management and virtualization services from a tile, which is considered unreliable, to a more reliable tile. In order to support different requirements regarding safety, timing integrity and minimized overhead for the relocation services, several established strategies can be enacted by the system management. The migration protocol for signaling during run configuration and actual relocation allows migration with minimal downtime and no communication loss. The actual migration is triggered by a configurable threshold on critical system parameters on a per task basis.

1 Introduction

The growing transistor integration densities following Moore's Law enabled manycore platforms with tens and even up to hundreds of processing cores per chip. So-called tiled architectures, where compute, IO and memory resources are structured in individual tiles interconnected by a packet-based Network-on-Chip (NoC), are a particularly promising set-up for scalable manycores (see Fig. 1 for a generic example of a tiled multicore). Existing implementations include Intel's "Single-Chip Cloud Computer" (SCC) [GHKR11] or the TILEPro100 platform by Tilera [Aga07], containing 24 and 100 tiles, respectively.

However, technological progress is not the only driver towards manycore architectures. The trend is to consolidate multiple applications, each consisting of several tasks with individual safety, security and real-time requirements onto a single shared processing platform.

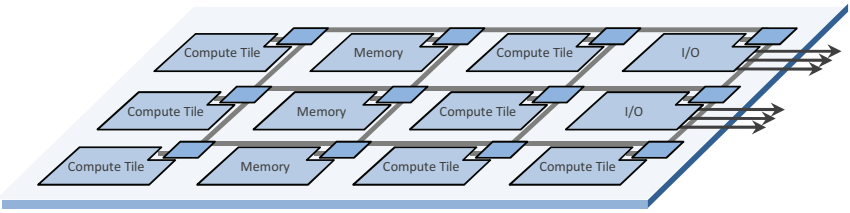

Figure 1: Generic tiled manycore architecture

Examples are sensor or media processing applications, augmented reality and robotic control or other recognition, mining and synthesis (RMS) applications [Dub05], all running on the same multi-/manycore processing platform. Those applications have in common that they require high performance and parallel execution.

The aforementioned platforms are subject to dependability issues as they favor latest technology implementation and 3D integration with increased potential error rates, either transient or permanent [B+04]. One example is stress by thermal hotspots resulting in intermittent errors in signal integrity or run time behavior (short term effects) as well as in physical damages in form of transistor aging or even electromigration [NX06]. To guarantee the operativeness of the applications, counter measures must be taken at all abstraction levels of integrated circuit and system design.

In this concept paper, we propose to enhance regular structured manycore architectures with HW supported processing and communication virtualization techniques in order to increase the fault tolerance of the applications. This is centered around the dynamic re-use of temporarily unused or weakly loaded tiles by transparent task relocation. The execution of a task is switched over from a tile marked as unreliable to a more reliable tile as a service by the underlying distributed system management. In order to support different requirements regarding safety, timing integrity and minimal relocation overhead, several established strategies can be enacted by the system management. tiles by

The remainder of the paper is structured as follows: In chapter 2 the platform with its basic building blocks and supporting extensions for virtualization is described. Chapter 3 explains the different error and failure concepts on this platform and the protocols to signal migration, triggered by task specific thresholds violations. The paper concludes with a summary in chapter 4.

2 Tiled Manycore Platform

2.1 Basic Building Blocks

Fig. 2 depicts a common tiled manycore platform constructed from a limited set of building blocks.

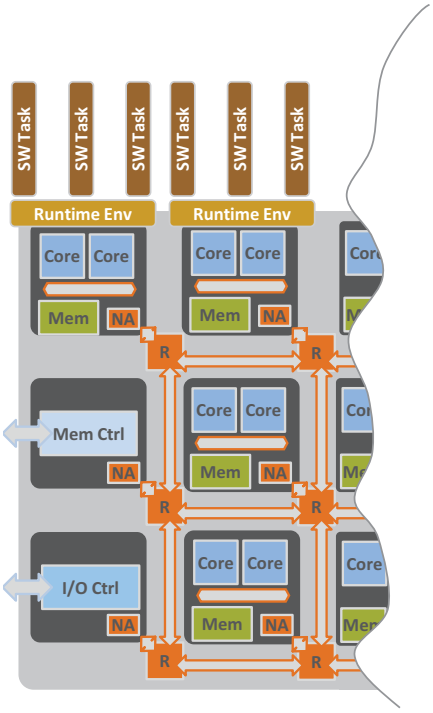

Figure 2: Tiled manycore architecture in detail

Processing is performed in compute tiles, consisting in general of one or few small RISC cores and local memory connected via a shared bus. Due to resource constraints and the overall design decisions compute tiles only have small feature set. To avoid complex cache coherence algorithms and synchronization, the basic runtime environment usually spans only individual compute tiles. It provides SW tasks running on top of it with limited services.

A NoC with router nodes and links acts as a generic communication infrastructure to connect individual tiles. Every tile has at least one network adapter (NA) providing access to the NoC. All interaction and data exchange between tiles is done via explicit communication, i.e. message passing, over the NoC. In order to prevent message loss or blockage in case of broken NoC routers or links, the NoC has to flexibly provision alternative routes.

IO, e.g. network interfaces or high-speed links, is encapsulated in own tiles and can is shared via the NoC by a large number of compute tiles. Memory resources are either on-chip, dedicated SRAM based memory tiles, or for larger memory ranges, off-chip DDR SDRAM modules accessible through memory controllers. Local memories within the compute tiles, on-chip SRAM tiles and off-chip memory form the memory hierarchy of the manycore.

The regular structure of such an architecture and the high number of available homoge-

neous (compute) tiles enables approaches to enhance the reliability by using (temporarily) unused tiles as fallback alternatives in case of failures and errors within the designated tiles. In general, limiting factor is the static configuration/deployment of the architecture and (up to now) missing methods and strategies to migrate tasks in a transparent way, i.e. without modifications and side-effects for the tasks and their runtime environment.

2.2 Extensions for Virtualization

As tiled manycore architecture provide a consolidating environment for hundreds or even thousands of tasks, those can have different requirements towards their running environments in terms of real-time, required operating system and performance/throughput. The system needs to be provisioned, scheduled and partitioned for those concurrently running tasks and environments.

Virtualization of the physical HW by a hypervisor or virtual machine monitor (VMM) is an established concept for providing such a function ([BDF+03], [Hei09]). Different domains are running in separate virtual machines (VM). A domain can include a single task with only a rudimentary operating system up to a complex general operating system with several tasks. The underlying platform resources are shared and compartmentalized by the hypervisor.

To avoid a single point of failure a hypervisor instance should be only controlling one or a few cores ([lin]) and communicate/coordinate with the the other hypervisor instances via a distributed system management and their local slaves.

To reduce the overhead for processing virtualization and enhance compartmentalization HW support for this is preferred. Similar approaches exist already for High Performance Computing (HPC), e.g. VT-X on Intel CPUs [NSL+06] and can be scaled down for cores in compute tiles. The explicit communication between compute, memory and IO tiles is highly performance critical. Here, dedicated HW support is also required to eliminate overhead otherwise occurring in software to provide virtualization for communication. To avoid changes to the NoC and and enable re-use of existing implementation this HW support should be added to the edges of it, i.e. within the network adapters in the tiles. As NoC is is packet-based concepts from the HPC network and IO virtualization can be adapted for this ([WSC+07]). IO tiles share the most resemblance to normal virtualized network interfaces and therefore have an extensive virtual network interface controller (VNIC). Virtual Network Adapter (VNA) for the compute tiles are a subset of the features of VNIC as their have a only a limited number of connections to share and to service.

Fig. 3 highlights the differences between a common tiled manycore architecture and one with virtualization extensions described above.

In order to achieve a performing and cost efficient realization for the VNIC/VNA entities, NoC packet queues, caches for individual communication configurations and packet buffer management should not be stored entirely in the respective entity. Although this would be advantageous from throughout and real-time aspects it essentially duplicates memory requirements. With the assumption that only a limited number of real-time and high-

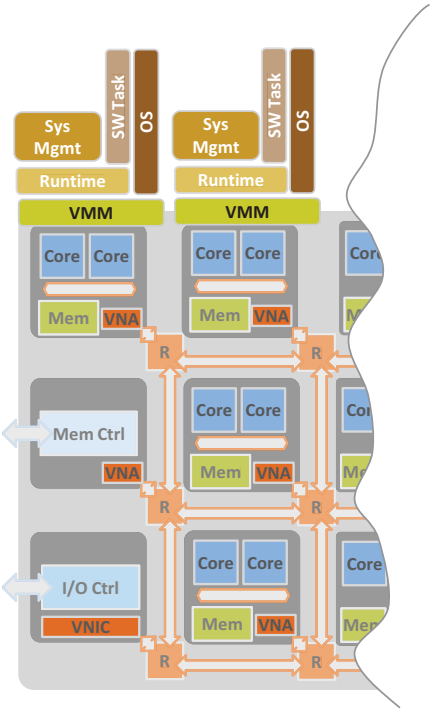

Figure 3: Tiled Manycore Architecture with virtualization extensions

priority connections are active at any point of time, it should be possible that a bounded number of queues/caches/buffers can be shared between those connections in a dynamic way [RWH10]. By managing those elements for real-time/high-priority tasks and for best-effort tasks by scheduling and (de-)multiplexing the associated resources, different levels of services can be provided (see Fig. 4.

Because of the incooperation of communication and processing virtualization it is now possible have transparent task relocation. Furthermore, hypervisors with their small footprint provide a minimal trusted computing base and can be used to enhance overall reliability even further via monitoring of runtime environments and their tasks [DKR08], via check-pointing complete runtime environments or via loose-lock-stepping ([TSKM08], [CLO+08]). This can be used to provide flexible (dual) modular redundancy for cores in intra and inter tile scope but without requiring special or dedicated HW. Drawback is a higher overhead due to handling it mostly in SW via hypervisor.

Results reported in the literature ([CCS10], [JRK10]) show only minimal downtimes during live migration of complete operating systems in HPC environment. We expect that for tightly integrated tiled manycores architecture with HW enablement for processing and communication virtualization with only small runtime environments live migration without communication data loss under real-time constraints should be possible.

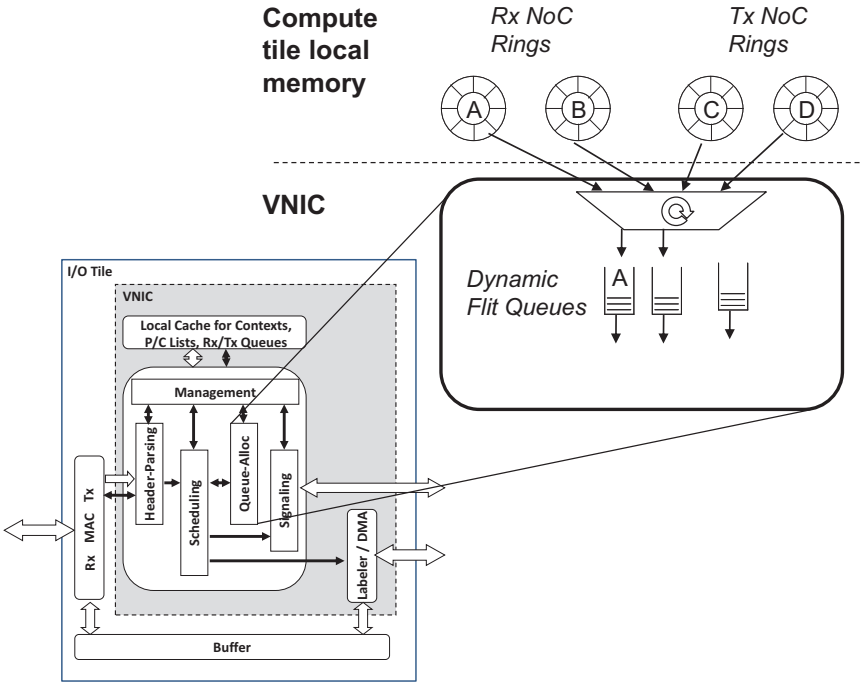

Figure 4: VNIC with dynamically shared queues for HRT and Best Effort communications

3 Error/Failure and Migration Handling

3.1 Classification

To decide if and when a task relocation is needed the components required for this task, e.g. runtime system, compute tile, cores, NoC or IO, have to be categorized by their operativeness. The following states are possible:

1. **No Error, No Failure**: The component is operating normally.
2. **Some Errors, No Failure**: The component is not operating error-free, but there is no failure yet. Errors include parameters exceeding a defined threshold, e.g. a too high temperature in a tile.
3. **Significant Errors, No Failure**: The component experiences errors over a defined rate, but this still does not result in a failure.
4. **Failure**: There is a failure and the component's behavior is corrupted.

The first three states can be handled by task relocation without information loss. The last state requires additional mechanisms to return to a valid state. According to varying

safety criticality and timeliness requirements, task relocation can be divided into different strategies based on existing protection methods [VPD04].

- **Cold Task Relocation**: No alternative set-up has been configured. The system management has to create this alternative from scratch. This involves reserving compute tile resources and NoC routes and activating them. This approach does not bind additional resources beforehand, but requires the most actions straight before and during the relocation. See Fig. 5a) for an example involving tasks on a compute tile processing IO data coming from an IO tile and and storing data on a memory tile.
- **Warm Task Relocation (1:1, N:1)**: An alternative set-up has been pre-configured. The basic system and task are existing, but not running. The state of the task has be transferred and the communication routes have to be activated. Overcommitting the alternative compute tile for different tasks is possible. This strategy should provide a good trade-off between increased reliability and utilized resources. See Fig. 5b) for an example.
- **Hot Task Relocation (1+1)**: An alternative location has been pre-configured and is running. All communication is already up both for the original and alternative location. In case of a relocation only the active setting is switched over. In this variant the task itself can be allowed to trigger its relocation and inform the system management afterwards to eliminate it from the critical decision path. Here, low latency, minimal downtime and high fault tolerance is achieved with high resource utilization. See Fig. 5c) for an example.

According to the criticality for an individual task, a particular strategy for task relocation is pre-defined during design time and potentially pre-configured by the system management during run time. This strategy is triggered when the task is under a dependability threat. Sensors within the tiled manycore architecture monitor the components and report failures and errors to the system management. This uses the aggregated information to generate a model of the system. See chapter 3.2 for the triggering and chapter 3.3 for the used protocol.

For cold and warm task relocation a chain of strategies can exist as to prior relocation of other tasks the first fallback is already impossible. Set-ups must be constructed and chosen carefully to prevent overload in the system in case of a migration.

A special case is task migration within a tile. This is only possible if this tile contains two or more cores. Such a migration is preferred due to the limited configuration effort in opposition to inter tile migration. No changes or reconfiguration other than within the tile have to be performed and no complex migration protocol is required (see below). Nevertheless, inter tile task migration is needed if the reason for the compute tile becoming unreliable are errors or failures in required components, e.g. the shared bus. Another reason is too strained resources in this tile to support a consolidation of all tasks on a reduced number of cores. Furthermore, task migration to another tile can be required because of errors and failures in the vicinity of the tile, e.g. existence of a thermal hotspot or problems with routers or network adapters nearby.

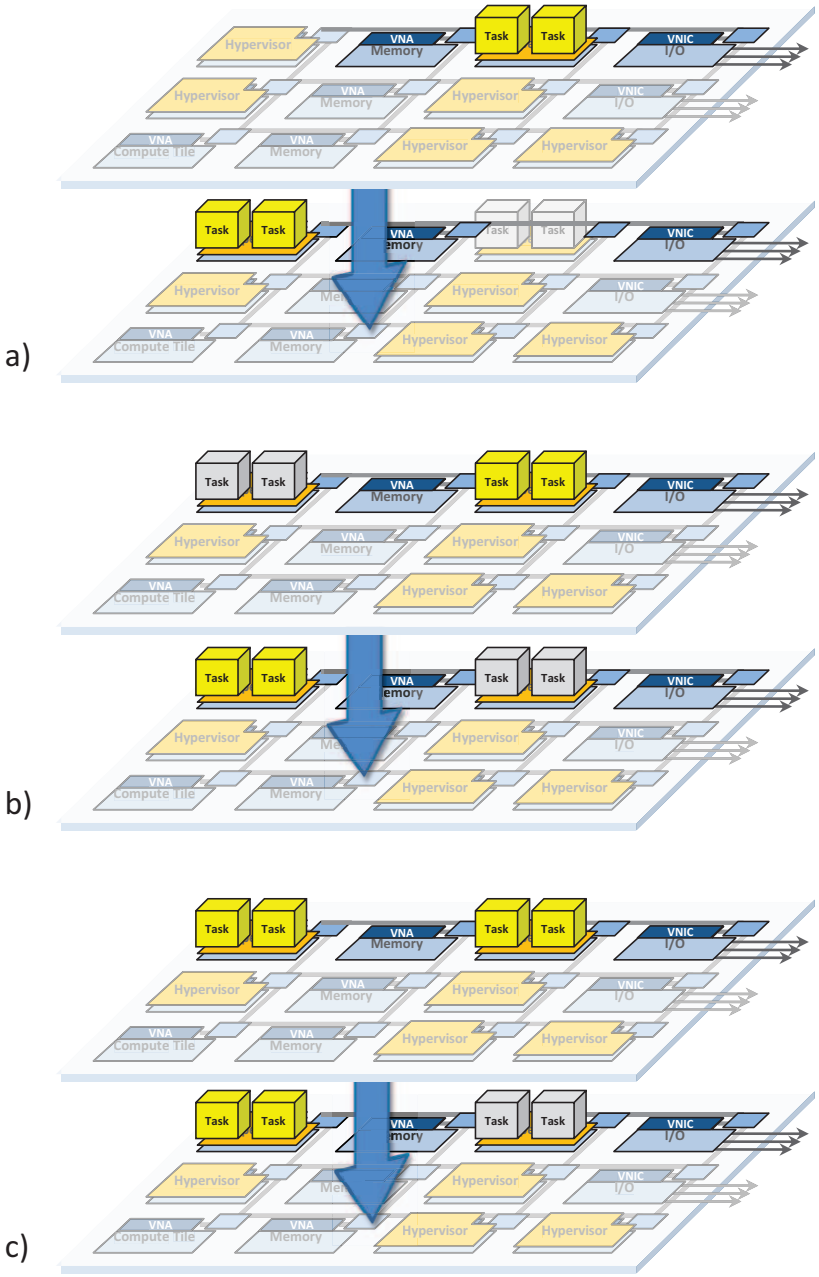

Figure 5: Task migration with different strategies a) cold task relocation, b) warm task relocation, c) hot task relocation involving two tasks on a compute tile processing IO data coming from an IO tile and and storing data on a memory tile

3.2 Threshold-based Migration

In the system management a pre-set matrix for each task exists with its sensitivity to error and failure states in a definable time slot and per individual components. Some tasks have a high sensitivity to errors and failures in certain components because this will almost certainly lead to a failure of the task, e.g. a wrong timer value for a real-time task. Other tasks are maybe more relaxed regarding errors or failures, for example small data corruption in video input data for a object recognition task can be ignored due to the noise robustness.

This matrix is weighted with factors decided during design time and used in the decision to relocate based on configurable threshold for this specific task. The criticality of the task is accounted for in the weighting, i.e. highly critical tasks have high weighting factors. The threshold is defined by the migration cost for this task. The relocation consists of two factors: The cost for the actual migration operation (the amount of to be transferred task data for this relocation between tiles in form of a memory snapshot, reconfiguration overhead) and the load increase of the system after migration in the new set-up (e.g. traffic over more NoC links). If the actual value generated by errors and failures for the weighted matrix is over the threshold the task relocation is triggered as configured by the respective system management slave. This concept is visualized in Fig. 6. This approach goes beyond Polze et. al [PTS11] which ignores the cost for migration and only triggers migration on a failure prediction.

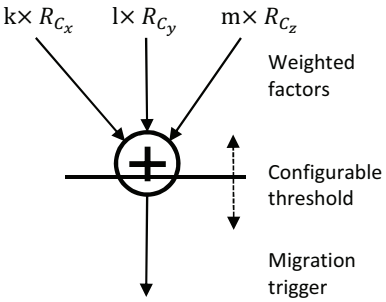

Figure 6: Visualized migration trigger calculation with configurable threshold

3.3 Protocol for Migration Management

As migration management has to act in a timely fashion we propose to integrate the critical steps of it as dedicated HW blocks and queues into the VNA and VNIC modules. The communication for the migration protocol has a higher priority and/or reserved communication resources in the NoC compared to the normal data communication. The system management can pre-configure alternative (i.e. shadow) set-ups. Such a set-up consists of a bundle of communication and processing configurations which have to exist for a task deployment. By deactivating the old set-up and activating a new set-up a task migration is

performed.

Each set-up has a global unique ID. This is used to locate the set-up (local memory in a tile or in a memory tile, for example by a VNIC). Linked to a set-up is a list of all involved tiles. Each tile has its own configuration linked to this set-up (e.g. the run time and hypervisor parameters and memory address ranges).

If triggered by a task threshold violation the responsible system management slave (SMS) initiates switching to the designated alternative set-up. The first step is writing the ID of the old set-up to be purged and the ID of the new set-up replacing it to a special configuration interface of the tile's VNA. Receiving this write the VNA locks itself against other task migration requests. It then transmits a request to VNAs and VNICs listed in the new set-up informing about the activation initiation of the new set-up.

VNAs receiving this request are locking themselves, are acknowledging it by replying with a response and are informing their respective SMS via notification. The SMS then starts reconfiguring its tile resources for this new set-up. After successful reconfiguration it then informs its VNA about it. The VNA sends then a second acknowledgment response to the triggering VNA.

VNICs receiving this request are locking themselves and are acknowledging it by replying with a response.

If all addressed VNAs and VNICs respond in a configurable time frame with all acknowledgment responses the migration protocol continue with the deactivation initiation of the old set-up by sending a request to the VNAs and VNICs listed in the old set-up. The VNAs and VNICs receiving this request are acknowledging it by replying with a response and locking themselves.

If all addressed VNAs and VNICs respond in a configurable time frame with the acknowledgment the switching is performed (including a downtime for the task). In this stage also running state information is exchanged in case of warm standby relocation.

The triggering VNA sends out a request for invalidating the old set-up. The addressed VNAs reply with an acknowledge response and inform their SMS about it which will then purge all related resources to the old set-up. For VNA which are both involved in the new and old set-up incoming messages are buffered for the time being. Otherwise communication is dropped.

VNICs involved reply with an acknowledgment response and if both active in the new and old set-up buffer communication during this downtime and drop them otherwise.

By adding an individual timer value per VNA/VNIC allows a tasteful shutdown of communication so that no communication will be dropped.

When receiving all the acknowledgment responses the triggering VNA sends out the activation for the new set-up. All VNAs and VNICs are using directly the new set-up. If communication is already incoming for a not yet activated set-up on a VNA or VNIC it is buffered until activated.

Sending out the deactivation of the old set-up and activation of the new set-up can be be done together if packet loss during the migration phase is acceptable.

Further requests for a different task migration are denied with a negative response message. Then the locking of the VNA/VNIC resource is acknowledged and also the system management part for this VNA is informed. It performs the needed resource requirements for task migration. If there is any time out or problem during the stages of the migration protocol, e.g. a VNA is already blocked for different and still running migration the process is stopped for now, the triggering VNA is informing all participant which roll back possible configuration changes.

A sequence of set-ups is possible to prevent deadlocks due to resource constraints, e.g. first a set-up is enacted to free up resources then the set-up for the actual task migration is done. Parallel task relocations are possible as long they do not share tiles with VNAs and VNIC to be reconfigured.

4 Summary

In this paper we have described how the regular structure of tiled manycore architectures can be utilized to enhance dependability of applications running on it. With HW supported communication and processing virtualization it is possible to migrate tasks transparent within such a platform under real-time and performance constraints. Different task migration strategies can be used based on the required reliability level of its application. For triggering the actual migration a threshold-based configuration on individual task basis is proposed. This concept would allow to run dependable applications on a platform with generally undependable components by preventing single points of failure, live substitution of failed components and virtual redundancy.

5 Acknowledgments

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program “Dependable Embedded Systems” (SPP 1500 - spp1500.itec.kit.edu).

References

[Aga07] A. Agarwal. The Tile processor: A 64-core multicore for embedded processing. In *Proceedings of HPEC Workshop*, 2007.

[B+04] S. Borkar et al. Microarchitecture and design challenges for gigascale integration. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–3. IEEE Computer Society, 2004.

[BDF+03] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[CCS10] F. Checconi, T. Cucinotta, and M. Stein. Real-time issues in live migration of virtual machines. In *Euro-Par 2009–Parallel Processing Workshops*, pages 454–466. Springer, 2010.

[CLO+08] K. Chanchio, C. Leangsuksun, H. Ong, V. Ratanasamoot, and A. Shafi. An efficient virtual machine checkpointing mechanism for hypervisor-based HPC systems. In *Proc. of the High Availability and Performance Computing Workshop (HAPCW)*, 2008.

[DKR08] T. Distler, R. Kapitza, and H.P. Reiser. Efficient state transfer for hypervisor-based proactive recovery. In *Proceedings of the 2nd workshop on Recent advances on intrusiton-tolerant systems*, page 4. ACM, 2008.

[Dub05] P. Dubey. Recognition, mining and synthesis moves computers to the era of tera. *Technology@ Intel Magazine*, pages 1–10, 2005.

[GHKR11] M. Gries, U. Hoffmann, M. Konow, and M. Riepen. SCC: A Flexible Architecture for Many-Core Platform Research. *Computing in Science & Engineering*, pages 79–83, 2011.

[Hei09] G. Heiser. Hypervisors for consumer electronics. In *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*, pages 1–5. IEEE, 2009.

[JRK10] B. Jiang, B. Ravindran, and C. Kim. Lightweight live migration for high availability cluster service. *Stabilization, Safety, and Security of Distributed Systems*, pages 420–434, 2010.

[lin] Hardware-Assisted Reliability Enhancement for Embedded Multi-core Virtualization Desig.

[NSL+06] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), 2006.

[NX06] V. Narayanan and Y. Xie. Reliability concerns in embedded system designs. *Computer*, 39(1):118–120, 2006.

[PTS11] A. Polze, P. Troger, and F. Salfner. Timely Virtual Machine Migration for Pro-Active Fault Tolerance. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, pages 234–243. IEEE, 2011.

[RWH10] H. Rauchfuss, T. Wild, and A. Herkersdorf. A network interface card architecture for I/O virtualization in embedded systems. In *Proceedings of the 2nd conference on I/O virtualization*, pages 2–2. USENIX Association, 2010.

[TSKM08] Y. Tamura, K. Sato, S. Kihara, and S. Moriai. Kemari: virtual machine synchronization for fault tolerance. In *Proceedings of the USENIX Annual Technical Conference 2008 (Poster Session)*, 2008.

[VPD04] J.P. Vasseur, M. Pickavet, and P. Demeester. *Network recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann Publishers, 2004.

[WSC+07] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A.L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 306–317. IEEE, 2007.