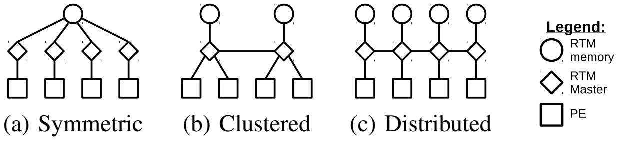
AGAMID: A Transaction-Level Framework for Design Space Exploration of Hardware-Enhanced run-time Management for Many-Core Processors

Daniel Gregorek, Alberto Garcia-Ortiz
Integrated Digital Systems Group, ITEM, University of Bremen, Germany
{gregorek,agarcia}@item.uni-bremen.de

Abstract— **The emergence of many-core processors raises novel demands to system design. Power-limitations and abundant parallelism require for efficient and scalable run-time management. However, the design of a many-core run-time manager generally suffers from exhaustive evaluation time. AGAMID is a novel research framework for design space exploration of hardware-enhanced many-core run-time management. The framework uses SystemC/TLM for flexible and productive system modeling while allowing evaluation at the timescale of seconds. We provide a generic run-time manager to compare arbitrary management systems and HW/SW partitioning. Real-world application workloads are applied by means of compact task graphs. The integrated system-call interface enables fast and task-accurate full-system simulation.**

I. Introduction

The many-core domain exacerbates the run-time management (RTM) complexity and demands for scalable and high-responsive implementations. The integration of dedicated hardware for RTM is therefore predicted to become mainstream for many-core systems [3]. However, there exists a large number of RTM architectures (see Fig. 1) and HW/SW partitionings, which makes design space exploration a time-consuming task.



(a) Symmetric (b) Clustered (c) Distributed

Fig. 1: Design point examples for RTM architectures. RTM components can either be HW or SW.

II. AGAMID Framework

AGAMID is a TLM design space exploration framework for comparison of many-core systems with an emphasis on the RTM architecture [1]. The framework permits design point evaluation at the timescale of seconds. Applications are abstracted as TLM traces while the RTM is simulated at a higher level of detail. The framework uses SystemC / TLM 2.0 for system modeling and a template-based generic RTM. The RTM applies a generic RTM master (see Fig. 2) to compare different HW/SW partitionings and communication models while using a common set of elementary functions.

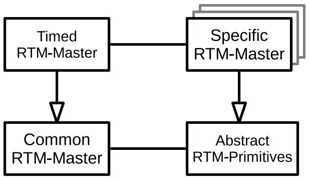
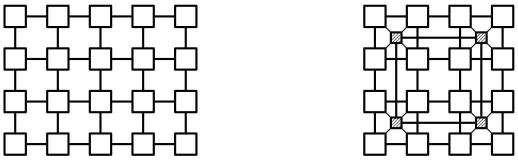


Fig. 2: UML class diagram for a generic RTM master

AGAMID has the capability to model a complete HW infrastructure for RTM [2]. Fig. 3 shows a conceptual comparison of a homogeneous many-core versus a many-core with dedicated RTM infrastructure. The dedicated RTM consists of components for computation and communication.

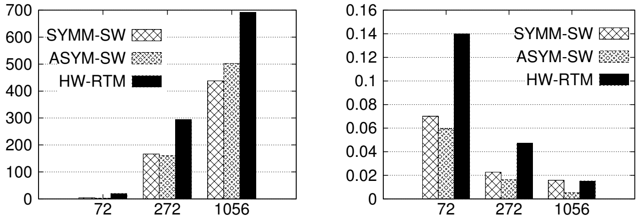


(a) Homogeneous (b) Dedicated infrastructure

Fig. 3: Conceptual comparison of (a) homogeneous many-core versus (b) many-core with dedicated RTM infrastructure.

III. Live Demo

Our demo presents hands-on results for many-core design space exploration. We compare RTM architectures, algorithms and HW/SW partitionings while using a variety of benchmarks. Fig. 4 exemplifies evaluation results for a hardware-enhanced run-time manager (HW-RTM) compared to a symmetric and an asymmetric software RTM.



(a) Application Speedup (b) Wall Time per core [s]

Fig. 4: Evaluation of simulated application speedup and wall time per simulated core. Fig. 4a indicates the advantage of hardware-enhanced run-time management while Fig. 4b demonstrates the scalability of the AGAMID framework for many-core simulation.

AGAMID is open source software and available from our website: http://www.ids.uni-bremen.de/agamid.html

References

[1] D. Gregorek and A. Garcia-Ortiz. A transaction-level framework for design-space exploration of hardware-enhanced operating systems. In *International Symposium on System-on-Chip (SOC 2014)*. IEEE, 2014.

[2] D. Gregorek and A. Garcia-Ortiz. The DRACON embedded many-core: Hardware-enhanced run-time management using a network of dedicated control nodes. In *International Symposium on VLSI (ISVLSI)*, 2015.

[3] V. Nollet, D. Verkest, and H. Corporaal. A safari through the mpsoc run-time management jungle. *Journal of Signal Processing Systems*, 60(2):251–268, 2010.