A Transaction-Level Framework for Design-Space Exploration of Hardware-Enhanced Operating Systems

D. Gregorek
Inst. of Electrodynamics & Microelectron., Univ. of Bremen, Bremen, Germany
Garcia-Ortiz, A.

Abstract

The increasing number of processing elements on embedded many-cores gives novel challenges for the chip design. Dedicated hardware has become an important feature to support the applied operating system and to improve the overall system efficiency. Since evaluation of novel architectures requires time expensive simulations or prototyping, transaction-level analysis gives an appropriate tool for early design stage evaluation. This work proposes a transaction-level framework for simulating hardware-enhanced many-core operating systems. The framework allows the design space exploration of the hardware and software architecture and uses a trace-based task description language including a customized interface for system calls.

Published in:
System-on-Chip (SoC), 2014 International Symposium on
Date of Conference: 28-29 Oct 2014
Page(s): 1 - 4
INSPEC Accession Number: 14790590
Conference Location : Tampere
DOI: 10.1109/ISSOC.2014.6972432
Publisher: IEEE

A Transaction-Level Framework for Design-Space Exploration of Hardware-Enhanced Operating Systems

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

***Abstract*— The increasing number of processing elements on embedded many-cores gives novel challenges for the chip design. Dedicated hardware has become an important feature to support the applied operating system and to improve the overall system efficiency. Since evaluation of novel architectures requires time expensive simulations or prototyping, transaction-level analysis gives an appropriate tool for early design stage evaluation.**

This work proposes a transaction-level framework for simulating hardware-enhanced many-core operating systems. The framework allows the design space exploration of the hardware and software architecture and uses a trace-based task description language including a customized interface for system calls.

***Index Terms*—transaction-level, hardware operating system, trace-driven simulation, design space exploration**

I. Introduction

The emerging many-core architectures demand for a scalable design of the applied operating system. The OS has to bring the dynamic requirements of the user applications into accordance with the monitored state of the chip. The demands for high-performance, low-power and deterministic computation time advice for hardware implemented solutions of the operating system. It is predicted that hardware support for run-time system management will attain into mainstream for many-core platforms [1]. Especially for embedded systems, where low-power is a crucial design constraint, the OS-specific implementation of hardware accelerators can become necessary.

To address the aforementioned challenges, different architectural approaches for the dedicated hardware of the operating system have been reported in the literature. Centralized approaches have been shown to have drawbacks in terms of the scalability for the on-line computation with an increasingly large number of cores [2]. Also, traffic hot-spots become a bottleneck for such designs. On the other hand, the overhead introduced by a fully-distributed approach can supersede the potential benefits [3]. As a consequence, choosing the right granularity for the dedicated on-chip hardware supporting the OS remains an insisting optimization problem.

The design space exploration for systems with many cores usually requires long simulations runs. It is therefore appropriate to apply an computationally efficient transaction-level model for architecture characterization at the early design stage [4]. Further, trace-based task description is a well known and long-used technique [5] to characterize the behavior of a user task in an abstract manner. This paper is about a transaction-level framework for the exploration of hardware-enhanced operating systems on many-core platforms and contains the following contributions:

1) A parametric simulation model for the exploration of a hardware-enhanced operating system architecture
2) A customized trace description language including a system-call interface

The remainder of this paper is organized as follows: In Section II we describe the implemented transaction-level simulation model. In Section III we present our customized trace-description language. Section IV shows experimental results, Section V discusses related work, and finally Section VI concludes the paper.

II. System Model

Our approach for the improvement of parallel computing performance is the application of a dedicated hardware infrastructure for operating systems. We therefore work on the development and evaluation of an on-chip network to accelerate the operating system services. The network uses message passing for communication between the dedicated OS hardware nodes. The messages have a header and one or more 32-Bit data fields. Fig. 1 displays the structure of a message. The header contains the message type (`type`), at least the source address (`src`), the priority (`prio`) and a broadcast flag (`flag`). The actual size of the message header depends on the message type and the hardware configuration (i.e. number of nodes/address width).

Figure 1: Message structure: Header + Data

We implemented a parametric SystemC model based on the TLM-2.0 library. An exemplary parametric architecture is given in Fig. 2. A set of dedicated global management nodes is connected by a global interconnect. Each global node controls a cluster of local nodes. Each local node contains a local controller and a common processing element. The communication between the dedicated hardware nodes is strictly message based. Each message is encapsulated inside a TLM generic payload. As a baseline we assume a homogeneous many-core processor (PEs drawn hatched). A

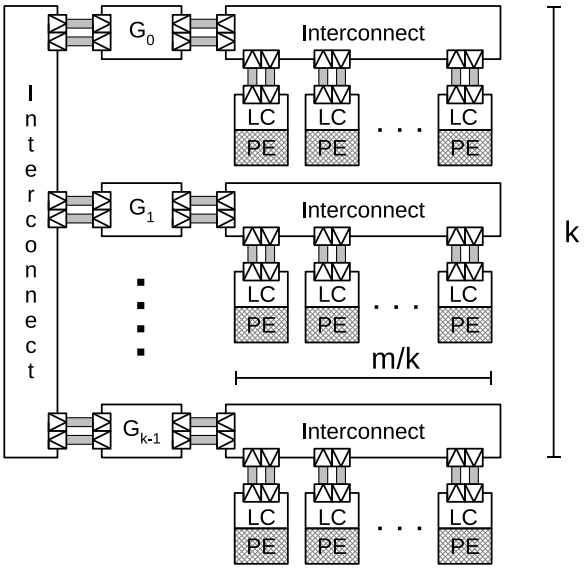

Figure 2: Outline of the parametric system model for the dedicated OS infrastructure having k global nodes (resp. clusters), and m processing elements.

common interconnect between the processing elements is left out for better readability. The model applies a transaction-level methodology using TLM-2.0 sockets and blocking transport. The simulated system has its own address space, although we partly use the address space of the host computer to achieve a higher simulation performance.

A. Global Nodes

The global nodes contain dedicated hardware for message transport, message handling, an OS-specific processor and private memory. Each of the global nodes runs one instance of an OS in software. Fig. 3 shows the structure for the global node. The software OS we use is loosely based on Micro-C/OS-II [6]. We extended the OS to have basic multi-core functionality and use a two-step task mapping algorithm. The first step maps a new task to a cluster, the second step maps to the local PE. Due to the complexity of the mapping decision we put special emphasis on that OS service. We model the delay t_s for one mapping step by means of Eqn. (1), where ν is the number nodes to be searched through and c_s is a timing parameter of our model. The $\mathcal{O}(\log \nu)$ search can be implemented e.g. by Red-Black Trees [7].

$$t_s = c_s \cdot \log \nu \tag{1}$$

During message handling and using the OS functionality, a computation delay is accumulated and charged to the nodes waiting time. We apply one simulation thread for each global node. The node operates non-preemptive, each message is processed completely before the next message is dequeued.

B. Local Nodes

The local nodes are constituted by a local controller and a processing element (PE). Fig. 4 shows the structure of a

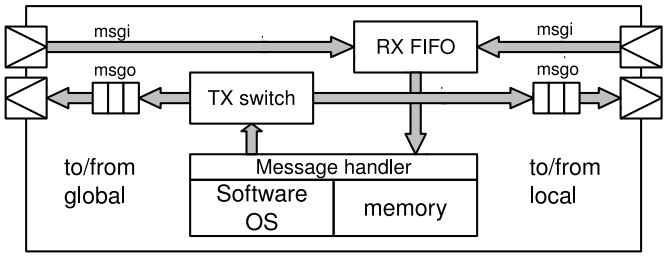

Figure 3: Structure of a global node with message queues, message transmitter and receiver, a dedicated OS processor and local memory

local node. The local node is controlled by the messages from the global manager and can send backwards the system-calls from the user tasks. The controller instructs the PE to start a user task, or to do a context switch, by updating the stack-pointer (`stk`). The PE is further controlled by the processor status register (`psr`) and the program counter (`pc`). For high simulation performance, the PE does not contain any general-purpose register, instead any register access by the user tasks is handled as an access to the address space of the host computer (see Sec. III). While the controller may stop the processor using the interrupt signal (`irq`), the processor can activate the controller using the trap signal (`trap`). The behavior of the processing element can be described by the following states:

- Idle: Wait for notification
- TDL: Execute trace of user task (see Section III)
- Trap: Signalize `trap` and wait

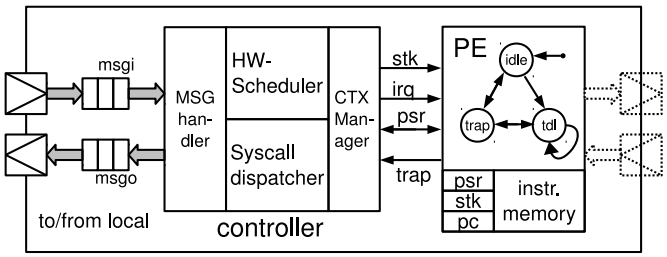

Figure 4: Structure of a local node with message handler and queues, Hardware-Scheduler, System-Call Dispatcher, Context-Manager and a processing element. Additionally, we indicate the interface to a common interconnect (dotted).

C. Interconnect model

In our current work the management interconnects are implemented as shared buses, but our framework allows to use other topologies as well. We extended the TLM generic payload with a priority field and a broadcast flag. For high simulation performance we apply a single priority queue for each bus interconnect to determine the next outgoing payload. The waiting time t_i for the TLM payload is approximated by Eqn. (2), where p indicates the specific priority of the payload. Broadcast messages are cloned and delivered to any connected target node.

$$t_i = [vol_{wait}(p) + vol_{mssg}] \,/ buswidth \tag{2}$$

III. TRACE DESCRIPTION LANGUAGE

To model the behavior of user tasks we developed a customized task/trace description language (TDL). The goal of the language is to provide a flexible possibility to abstract the characteristics of user applications including an interface for system calls. TDL is directly interpreted by the processing elements. A reference about the implemented TDL instructions is given in Tab. I. Vital element is the `tdl_wait` instruction which models the waiting time of the user task and therefore blocks the PE. Memory accesses also induce a waiting time, depending on the returned delay from the interconnect.

Table I: TDL instructions

Name	Args.	Description
`tdl_wait`	time	wait for given time
`tdl_sysc`	type + args.	call operating system
`tdl_memr`	addr, size	read from memory and wait
`tdl_memw`	addr, size	write to memory and wait
`tdl_calc`	in1, op, in2	arithmetic
`tdl_copy`	src, dst	copy content of register
`tdl_bnez`	value, addr	branch not equal zero

The system-call interface, address arithmetic and loop counters heavily depend on register accesses. Since the PE model does not contain user registers, TDL includes the capability to read and write into virtual registers. The register names are given by specific TDL labels shown in Tab. II. Each user task has its own context (resp. set of virtual TDL registers) which is stored inside the address space of the host computer. The register `tdl_reg_wait` is hidden to the user task and stores the currently charged waiting time of the task.

Table II: Virtual TDL registers

name	description
`tdl_reg_sysc`	System call type
`tdl_reg_arg0`	System call arg0
`tdl_reg_arg1`	System call arg1
`tdl_reg_sval`	System call return value
`tdl_reg_dptr`	Data pointer
`tdl_reg_loop`	Loop counter
`tdl_reg_wait`	charged wait time

The system-calls, which we apply throughout this paper are explained in Tab. III. We use a customized join/barrier mechanism to synchronize a parallel application. To reduce the run-time number of system-calls, a child task is allowed to exit immediately, when calling `os_join_exit`.

IV. EVALUATION

Our primary criterion for architecture evaluation is the throughput time (response time) t_r of an parallel user application. We measure the speedup S as the ratio of the sequential throughput time $t_{r,seq}$ versus the achievable parallel throughput time $t_{r,par}$. Having n independent user tasks of equal length l and m processing elements, the maximal achievable speedup is limited by a temporal management overhead $\Omega(m,n)$ as shown in Eqn. (3):

$$S = \frac{t_{r,seq}}{t_{r,par}} = \frac{n \cdot l}{t_{r,par}} = \frac{n \cdot l}{\lceil n/m \rceil \cdot l + \Omega(m,n)} \qquad (3)$$

Table III: System calls

Name	in	out	Description
`os_prog_spwn`	imem, dmem		Spawn new user task with given instruction- and data-memory addresses
`os_prog_exit`			Terminate task
`os_join_init`	count	addr.	Initialize join barrier with given count and return address to user
`os_join_free`	addr.		Free join barrier from memory
`os_join_wait`	addr.		Let task wait until counter is zero
`os_join_exit`	addr.		Decrement counter and terminate task

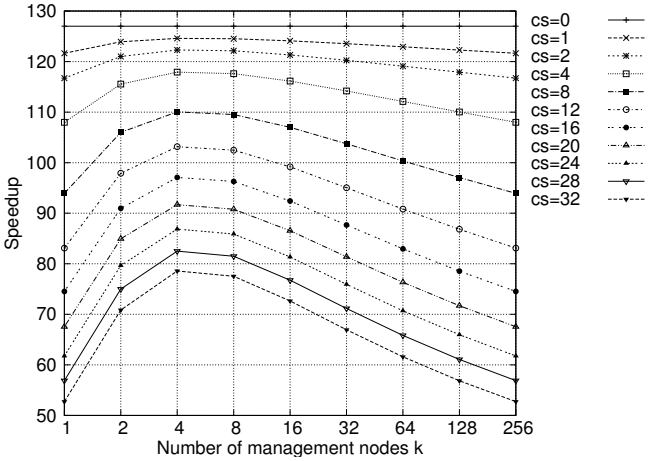

Figure 5: Analytic speedup model versus the number of global nodes k. Different values for the coefficient c_s are given for mapping $n = 127$ user tasks to $m = 256$ PEs

We approximate an overhead $\Omega \approx \Omega_{map}$ due to task mapping depending on m, n and the number of global nodes k. Assuming a temporal overhead given by Eqn. (1) for each step of our task mapping algorithm ($\Omega_{map} = t_{s,1} + t_{s,2}$), we constitute $\Omega_{map}(m,n,k)$ as the accumulated delay of the two mapping steps in Eqn. (4). The diagram of the expected speedup $S(m,n,k)$ using $\Omega = \Omega_{map}$ and different values for the coefficient c_s is shown in Fig. 5.

$$\Omega_{map}(m,n,k) = \overbrace{n \cdot c_s \cdot \log k}^{map\ global} + \overbrace{\frac{n}{k} \cdot c_s \cdot \log\left(\frac{m}{k}\right)}^{map\ local} \qquad (4)$$

In the following we want to test our analytic model by means of the transaction-level framework. Table IV gives the default parameters for our simulation model. We use the trace-description language TDL to model a synthetic parallel benchmark. In the experiment we include interference between two competing applications. The applications have an inter-arrival time λ, which is Poisson distributed and has a mean value of $\lambda = 7999$ Ticks. Each application contains $n = 127$ independent child tasks of equal length $l = 16000$ Ticks. The parallel tasks are synchronized using the system-calls given in Tab. III. The stimulus is injected directly into the message queue of a randomly chosen global node.

Table IV: Default Parameters

Name	Value
Number of processing elements m	256
Global bus width	32 bit
Local bus width	32 bit
Message receive delay	4 Ticks
Message transmit delay	4 Ticks
Simulation length	1e7 Ticks

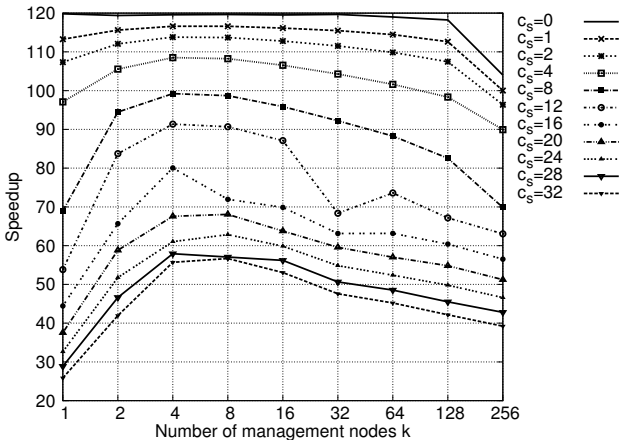

Figure 6: Measured speedup versus the number of management nodes k. Different values for the coefficient c_s are given for mapping $n = 127$ user tasks to $m = 256$ PEs

The quality of the measured speedup S in Fig. 6 fits to our analytic model and emphasizes the significant impact of the applied mapping algorithm to the overall system performance. The simulations confirm an optimum for the number of global nodes. The quantity of the measured speedup S is slightly smaller compared to the analytic model because of the additional OS communication overhead.

Values for the required simulation time, depending on the number of PEs and the number of stimulus iterations, are given in Tab. V. The simulations were run using an Intel-Celeron-M® processor at 1.40GHz having a 1024kB cache. Simulating one point in the design-space usually required less than a minute. A profiling analysis revealed that string parsing of the TDL instructions is a bottleneck for the simulation performance. As a follow-up, we plan to develop a TDL assembler and to compare our hardware-enhanced OS to a common software OS. We therefore extend the PE model and the trace description language TDL to include the capability to emulate a common software OS at the processing elements.

Table V: Simulation time in seconds. Each iteration of the stimulus contains $n = m$ child tasks

Number of PEs m	Iterations				
	1	10	20	40	80
64	0.11s	0.96s	1.98s	3.91s	7.56s
128	0.23s	1.98s	3.66s	7.04s	14.38s
256	0.51s	3.52s	7.46s	14.29s	32.68s
512	0.89s	7.68s	13.92s	29.61s	57.67s

V. Related Work

There exist various related works concerning the design-space exploration of hardware implemented operating systems. Muller et al. [8] describe a virtual platform for the hardware/software co-design of a real-time hardware operating system. Nexus++ uses an application specific circuit resolving time-critical task dependencies at run-time [9], and applies a trace-based description of a H.264 benchmark. A distributed and dedicated hardware approach has been implemented by Isonet [2]. Isonet applies a fully-distributed network of dedicated management nodes for hardware supported load balancing. According to the authors, Isonet uses its own cycle-accurate trace-driven simulator to evaluate a configuration of 1024 cores. However, to our best knowledge none of the related works provides a publicly available simulation framework, nor puts a focus on the description of the system-call interface.

VI. Conclusion

We presented a transaction-level framework for the design-space exploration of a dedicated hardware operating system. The OS runs parts of its functionality in software and is enhanced by a hardware-implemented message passing infrastructure. We provide a customized trace description language (TDL) including a system-call interface. For high simulation performance, parts of the simulated system run inside the address space of the host computer. We evaluated a proposed operating system architecture experimentally by means of a parallel benchmark and compared the measured performance of the benchmark against an analytic model. The framework is computationally efficient and allows to quantify the impact of the operating system architecture to the overall system performance at an early design-stage.

References

[1] V. Nollet, D. Verkest, and H. Corporaal, "A safari through the mpsoc run-time management jungle," *Journal of Signal Processing Systems*, vol. 60, no. 2, pp. 251–268, 2010.

[2] 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," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 1080–1093, 2013.

[3] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Exploration of mpsoc monitoring and management systems," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*. IEEE, 2011, pp. 1–3.

[4] T. Wild, A. Herkersdorf, and R. Ohlendorf, "Performance evaluation for system-on-chip architectures using trace-based transaction level simulation," in *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, vol. 1. IEEE, 2006, pp. 1–6.

[5] E. J. Koldinger, S. J. Eggers, and H. M. Levy, "On the validity of trace-driven simulation for multiprocessors," in *ACM SIGARCH Computer Architecture News*, vol. 19, no. 3. ACM, 1991, pp. 244–253.

[6] J. J. Labrosse, *Microc/OS-II*. R & D Books, 1998.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2.

[8] F. Muller, F. Muhammad *et al.*, "An embedded, generic and multiprocessor hardware operating system," *Design and Architectures for Signal and Image Processing (DASIP)*, 2009.

[9] T. Dallou and B. Juurlink, "Hardware-based task dependency resolution for the starss programming model," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*. IEEE, 2012.