A Power-Aware Framework for Executing Streaming Programs on Networks-on-Chip [*]

Nilesh Karavadara, Simon Folie, Michael Zolda, Vu Thien Nga Nguyen, Raimund Kirner
School of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
{n.k.karavadara, s.folie, m.zolda, v.t.nguyen, r.kirner}@herts.ac.uk

Abstract

Software developers are discovering that practices which have successfully served single-core platforms for decades do no longer work for multi-cores.

Stream processing is a parallel execution model that is well-suited for architectures with multiple computational elements that are connected by a network.

We propose a power-aware streaming execution layer for network-on-chip architectures that addresses the energy constraints of embedded devices.

Our proof-of-concept implementation targets the Intel SCC processor, which connects 48 cores via a network-on-chip. We motivate our design decisions and describe the status of our implementation.

1 Introduction

Software developers are discovering that practices which have successfully served single-core platforms for decades do no longer work for multi-cores. Different software engineering practices, programming paradigms, and execution models are needed to get the best of performance out of new parallel architectures, while obeying the energy constraints of embedded and the thermal constraints of high-performance devices.

Dataflow programming [9] is a particularly promising model for concurrent programming. Unlike most traditional programming languages—which are centered around control flow—data flow languages expose concurrency directly through explicit modelling of data dependencies. *Coordination languages* [3] allow software engineers to build parallel applications from sequential building blocks. *Stream processing* [14] is a parallel execution model that is well-suited for architectures with multiple computational elements that are connected by a network. Put together, these mechanisms afford a powerful software development approach for multi-cores [5, 4, 13].

We propose a power-aware streaming execution layer for network-on-chip architectures that addresses the energy constraints of embedded devices. We focus on the following requirements:

- Running on a top of a host operating system that already provides core services such as memory management and threading.
- Providing a uniform presentation of distributed computing resources, offering transparent task management, inter-task communication and task migration services across distributed cores and memory.
- Providing profiling and monitoring of computational resources, memory, and power consumption.
- Providing control over power management and frequency scaling.

After reviewing the LPEL scheduling layer (cf. Section 2), the SCC architecture (cf. Section 3), and the SCC power features (cf. Section 3.1), we present a design for a distributed variant of LPEL (cf. Section 4). We conclude after a review of related work (cf. Section 5).

2 LPEL

The *Light-Weight Parallel Execution Layer* (LPEL) [13] is an execution platform for stream processing on shared memory architectures. It provides user-space threading and communication mechanisms, building on operating system

[*]The research leading to these results has received funding from the FP7 ARTEMIS-JU research project "ConstRaint and Application driven Framework for Tailoring Embedded Real-time Systems" (CRAFTERS). under contract no 295371. The research was further supported by the Material Transfer Agreement 2010-2013 for the Intel SCC Research Processor "Light-weight Parallel Execution Layer for Stream Processing Networks on Distributed Memory Architectures".

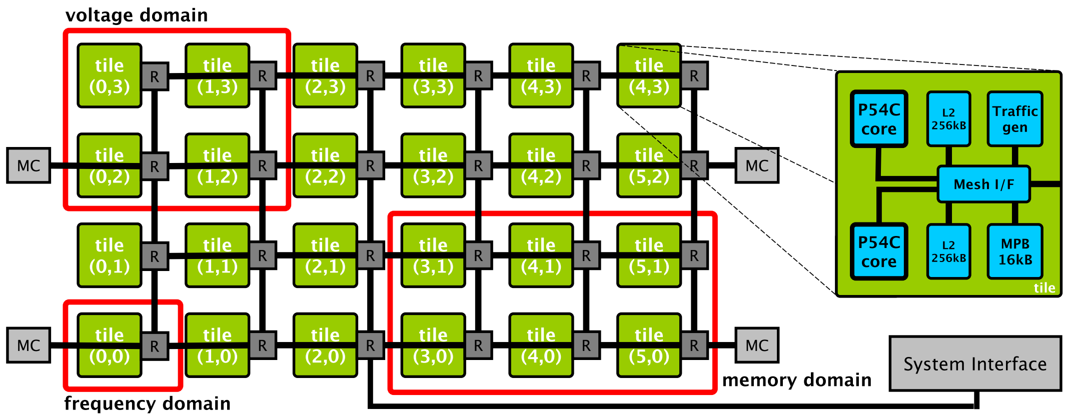

Figure 1. SCC Top-Level Tile Architecture

services such as kernel-level threading, memory management, atomic instructions and time-stamping.

LPEL uses the *portable coroutine library* (PCL) [11] to provide cooperative user level threads (ULT) over kernel level threads (KLT). PCL offers a light-weight, low-level mechanism for user-space context-switching, a feature from which applications with a large number of threads benefit.

LPEL maintains a set of workers running on separate kernel-level threads which are exclusively pinned to individual cores. These workers manage disjoint sets of tasks (ULTs). Each stream processing node instantiates a task with its own execution stack. Tasks are assigned to individual workers at their creation time and are scheduled locally.

For inter-task communication, uni-directional streams—implemented as a bounded buffer FIFO—are used. The state of a task changes according to the availability of the input and output items. Reading from an empty stream or writing to a full stream causes the task to be blocked.

A mailbox is a multiple-producer single-consumer queue that allows for concurrent enqueueing and dequeueing operations. Mailboxes are used for inter-worker communication. Workers wait until a message arrives into the mailbox, unless there is some ready task to be executed.

3 SCC Architecture Overview

The Intel *Single-Chip Cloud Computer* (SCC) has 48 cores connected via a network-on-chip [7]. As illustrated in Figure 1, the SCC consists of a 4x6 tile grid connected by a high bandwidth, low latency, 2D mesh network with hardwired X-Y routing. Four memory controllers (MC) support a total of 16 to 64 GiB external DRAM. Each tile contains two modified P54C processor cores with 32 KiB L1 cache, 256 KiB L2 cache, a 8 KiB SRAM *message passing buffer*, and a router.

The programmer has to maintain memory coherency explicitly. For low latency, the MPB is used to move L1 cache lines between cores, bypassing main memory. A special *message passing buffer type* (MPBT) tags cache lines for shared data in L1. Tagged data bypasses the L2 cache and goes directly into L1 cache. The instruction set architecture (ISA) has been extended with an instruction to invalidate all changed cache lines in L1 tagged as MPBT. Access to this invalidated L1 cache lines forces an update of the L1 cache lines with the data in the shared memory.

3.1 SCC Voltage and Frequency Scaling

Each tile constitutes an island for which the frequency can be set between 100 and 800 MHz. The various controllers and the mesh also constitute their own frequency islands. The frequency of the mesh can be set to 800 or 1600 MHz. Moreover, 2x2 arrays of tiles constitute islands for which the voltage can be set between 0.7 and 1.3 V at a granularity of 6.25 mV. The mesh constitutes a seventh voltage island. All adjustments are under application control.

Although SCC is not an embedded processor it has many features in common with future embedded processors. As an example, NoC architecture and DVFS are common features in KALRAY's MPPA MANYCORE [1] and SCC. Thus, we believe that our approach can be used for future embedded processor as well.

4 Distributed LPEL

We describe the distributed light-weight parallel execution layer (D-LPEL), designed to run on top of tile-based network-on-chip (NoC) processors such as the SCC. Our design allows for simple and efficient power and performance management, deploying the frequency/voltage configurations the SCC provides for power management. Such a framework is useful for embedded systems with a high demand for performance as well as energy efficiency. In the following we describe the task management of D-LPEL and explain why our design is suitable for power management on streaming networks on NoC-based processors.

D-LPEL provides a unified interface for task management/migration, inter-task communication, load balancing, profiling, and monitoring. It exploits architecture-specific features like power/frequency scaling on SCC. Figure 2 shows the architecture of D-LPEL, which sits between OS and application layer, e.g., S-Net [5, 4], elevating services from kernel-space to user-space.

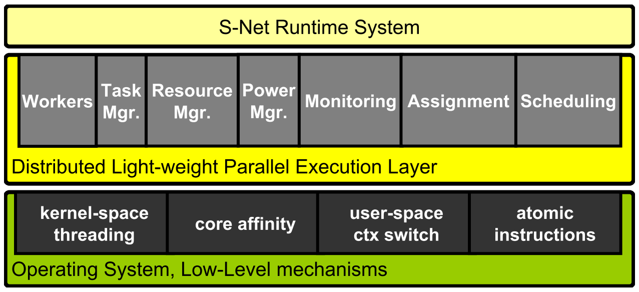

Figure 2. Architecture of D-LPEL with the runtime system of the S-Net coordination language running on top of it

D-LPEL extends LPEL with a power manager and a scheduler with support for load-balancing. D-LPEL is a distributed execution layer for NoC processors based on the multi-kernel paradigm. These kernels are called *workers* in D-LPEL terminology. Each worker runs on top of a single-kernel host operating system (OS). To control space scheduling and time scheduling, D-LPEL bypasses the scheduler of the host OS by a fixed mapping of a worker to one core. Program tasks in D-LPEL are scheduled at user level, with less overhead compared to scheduling different processes at the host OS.

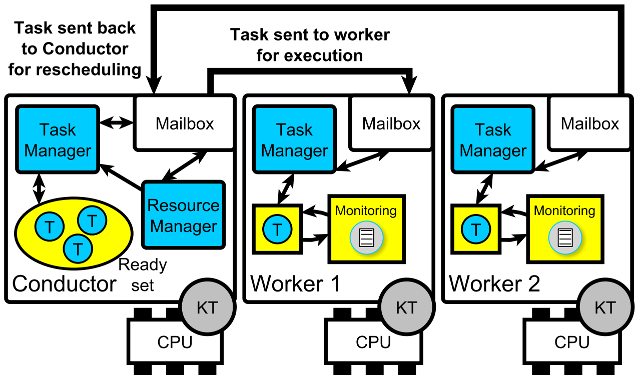

Figure 3. conductor worker correlation

As shown in Figure 3, the scheduling in D-LPEL follows a centralised approach, where one special worker—the so-called *conductor*—manages task queues and is responsible for distributing jobs to workers. The communication between conductor and workers is performed over mailboxes.

Task switching can cause overhead and may create performance bottlenecks. Execution time for task should be long enough such that task switching overhead becomes insignificant.

The built-in monitoring framework gathers task-specific resource usage information which in turn gets interpreted by the resource manager to aid the task manager in making scheduling and migration decisions. The monitoring information is also used to utilise architecture-specific features such as voltage and frequency control at system or domain level. These components work together to ensure system-wide load-balancing and maximum resource utilisation while fulfilling power and other constraints. Based on the message-based communication between conductor and workers, D-LPEL ensures portability across distributed-systems.

The assignment of ready tasks in D-LPEL to workers is done by the conductor. The selection of a task to be scheduled for execution is based on the performance demand at the individual streams connecting the tasks: The more messages are buffered at the output stream of a tasks, the lower its priority is. In contrast, the more messages are buffered at the input stream of a task, the higher its priority is. The scheduling heuristics are described precisely in [12]. Currently the conductor does not take into account the communication topology of streaming network.

Power/Energy Management By observing the input and output rates of the application or sub-networks, D-LPEL can detect power saving potential. Figure 1 shows the tile architecture of the SCC, including the voltage domains. By controlling the tile frequency and voltage of each voltage domain, D-LEPL can adjust the energy consumption of the SCC to the minimum value where the system can still cope with the performance demand.

D-LPEL detects whether a sub-network of the application currently is able to cope with the demanded input data rate by comparing the observed input and output rates. If the average output rate falls behind its profiled ratio to the input rate, D-LPEL increases the processor performance (voltage and frequency) to the level where the performance demand is satisfied.

By using the stream rates of streaming networks, D-LPEL's scheduling of streaming networks allows for an efficient power/energy management. For streaming networks with fixed multiplicity of messages from input to output this approach works by just comparing the input and output rates. For streaming networks with a variable multiplicity the system has to observe the average multiplicity in order to interpret the relation of input rate to output rate.

5 Related work

Verstraaten developed a port of the S-Net coordination language to the SCC [15]. In that approach tasks are mapped by static user annotations to a particular core. In contrast, our approach allows dynamic load balancing among cores.

Barrelfish is an open-source research multi-kernel operating system developed by ETH Zurich and Microsoft Research [2]. Barrelfish treats the machine as a network of independent cores and assumes no inter-core memory sharing. Barrelfish has been ported to the SCC, but it does not exploit its power management features.

MetalSVM [10] is a shared virtual memory management system comprised of a kernel and a hypervisor and developed at RWTH Aachen. The main goal of MetalSVM is to provide a cache coherent virtual machine where one instance of an OS runs on the SCC, creating the illusion of a cache coherent multicore system. To the best of our knowledge, MetalSVM does not tap into power management.

Phillip Gschwandtner et al. explore a wide range of system configurations—including different frequency/voltage settings—for various benchmarks[6]. This work exposes the effect of these settings, but it does not provide any framework for power management.

Ionnou et al.[8] present a power management approach for MPI applications. A runtime *phase predictor* adaptively partitions program execution into recurring *phases*. Whenever a phase is entered, the frequency for the respective core is readjusted based on previously observed execution times. The approach aims at an optimal performance/power yield by holding a set performance penalty. This is, however, not always feasible, because frequency changes can only be made at the granularity of tiles. This can cause problems with conflicting demands from different cores on the same tile. Our approach, on the other hand, can migrate tasks between power islands and therefore avoids the problem.

The majority of work we found on exploiting the power management functionality of the SCC makes use of the power management API of RCCE, a communication library inspired by MPI and specifically targeted at the SCC. In our approach we propose framework for a coordination language with implicit power management, based on the state of the streaming network.

6 Summary and Conclusion

In this paper we have proposed a power-aware streaming execution layer for network-on-chip architectures that addresses the energy constraints of embedded devices. Such a layer is useful as middleware for multi-core systems that are developed according to the dataflow programming and coordination approaches.

Although our work is still at an early stage, we believe that putting special emphasis on power-awareness, our proposal is an important step towards facilitating these approaches on systems that impose rigid power and thermal constraints, like embedded systems and high-performance devices.

References

[1] KALRAY MPPA MANYCORE Flyer, 2012. `http://www.kalray.eu/IMG/pdf/FLYER_MPPA_MANYCORE.pdf` accessed 13-Feb-2014.

[2] A. Baumann, P. Barham, P. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 29–44, 2009.

[3] P. Ciancarini and T. Kielmann. Coordination models and languages for parallel programming. In *Proc. Int. Conf. on Parallel Computing (PARCO99)*, pages 3–17. Imperial College Press, 1999.

[4] C. Grelck, S. Scholz, and A. Shafarenko. Asynchronous Stream Processing with S-Net. *International Journal of Parallel Programming*, 38(1):38–67, 2010.

[5] C. Grelck, S.-B. Scholz, and A. Shafarenko. A Gentle Introduction to S-Net: Typed Stream Processing and Declarative Coordination of Asynchronous Components. *Parallel Processing Letters*, 18(2):221–237, 2008.

[6] P. Gschwandtner, T. Fahringer, and R. Prodan. Performance analysis and benchmarking of the intel scc. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 139–149. IEEE, 2011.

[7] SCC External Architecture Specification (EAS). Technical report, Intel Labs, November 2010. Revision 1.1.

[8] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra. Phase-based application-driven hierarchical power management on the single-chip cloud computer. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 131–142. IEEE, 2011.

[9] W. M. Johnston, J. R. P. Hanna, and R. J. Millar. Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1–34, Mar. 2004.

[10] S. Lankes, P. Reble, C. Clauss, and O. Sinnen. The path to metalsvm: Shared virtual memory for the scc. In *The 4th symposium of the Many-core Applications Research Community (MARC)*, page 7, 2011.

[11] D. Libenzi. *GNU Portable Coroutine Library*, 2012. URL: `http://www.xmailserver.org/libpcl.html`.

[12] V. T. N. Nguyen and R. Kirner. Demand-based scheduling priorities for performance optimisation of stream programs on parallel platforms. In *Proc. 13th International Conference on Algorithms and Architectures for Parallel Processing*, LNCS, Sorrento Peninsula, Italy, Dec. 2013. Springer.

[13] D. Prokesch. A Light-Weight Parallel Execution Layer for Shared-Memory Stream Processing. Master's thesis, Technical University of Vienna, Vienna, Austria, 2011.

[14] R. Stephens. A survey of stream processing. *Acta Informatica*, 34(7):491–541, 1997.

[15] M. Verstraaten. High-level Programming of the Single-chip Cloude Computer with S-Net. Master's thesis, University of Amsterdam, 2012.