2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip

Comparison of Shared and Private L1 Data Memories for an Embedded MPSoC in 28 nm FD-SOI

Gregor Sievers*, Julian Daberkow*, Johannes Ax*, Martin Flasskamp*, Wayne Kelly†,
Thorsten Jungeblut*, Mario Porrmann*, Ulrich Rückert*
*Cognitronics and Sensor Systems Group, CITEC, Bielefeld University, Bielefeld, Germany
†Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia
Email: gsievers@cit-ec.uni-bielefeld.de w.kelly@qut.edu.au

***Abstract*—Tightly coupling CPUs within clusters allows for low latency, high bandwidth communication in MPSoCs. Our CoreVA-MPSoC integrates multiple clusters using an on-chip network. In this work we introduce a shared L1 data memory that can be accessed by all CPUs of a cluster with low latency. A Mesh-of-Trees (MoT) and a crossbar as topology for the shared memory interconnect are presented. A cluster that integrates the shared L1 memory is compared with an architecture that features an AXI interconnect and a local L1 data memory for each CPU. In addition, we consider an architecture that integrates both. We present implementation results using a 28 nm FD-SOI standard cell technology. The shared L1 memory shows similar area results compared to the local memory architecture. Place and route results of a cluster with 8 CPUs, 128 kB local- and 128 kB shared L1 data memory divided into 16 memory banks show a frequency of 728 MHz and an area of 1.77 mm^2. To map programs to the different CPU cluster configurations a compiler for streaming applications is used. An architecture with both local and shared L1 data memory and 4 memory banks shows best performance results in combination with a high resource efficiency.**

I. Introduction

Multiprocessor System-on-Chips (MPSoC) have become popular in many embedded markets due to their increased energy efficiency and computational performance compared to single CPU systems. More and more processing cores are being integrated on single chips due to the decreasing feature size of microelectronic circuits. In addition, the amount of on-chip memory continues to increase and occupies a significant portion of the overall MPSoC area. Traditionally, a memory hierarchy with several (private and shared) on-chip caches, external DRAM and a unified address space is used. This allows for easy programming, but results in unpredictable memory access times as well as high area and power requirements for the cache logic and coherence handling. Software-managed scratchpad memories can be used as a resource-efficient alternative to caches [1]. Most scratchpad memories are attached to a single CPU and CPU to CPU communication can be handled via a DMA controller that transfers data from one scratchpad memory to another. Recently, several multiprocessor architectures with shared multi-banked scratchpad memories have been proposed [2]–[5]. These architectures feature a low latency memory access time of one or two clock cycles when CPUs access the shared memory. Due to the uniform memory access, copying data can be omitted which results in high performance and high resource efficiency. The tight coupling of CPUs and shared memory requires a low latency, high performance interconnect.

Our hierarchical CoreVA-MPSoC architecture [6] features a Network on Chip (NoC) interconnect that couples several processor clusters. Within a CPU cluster several VLIW CPU cores are tightly coupled via a bus-based interconnect and share a common address space. This hierarchical approach combines the scalability of NoCs and high communication bandwidth within each CPU cluster. In [6] we compare several interconnect topologies (shared bus, crossbar, and NoC) and bus standards (AMBA AXI and OpenCores Wishbone) for 8 to 32 CPU cores. The CPUs can access each other's local level-1 (L1) scratchpad memory in a Non-Uniform Memory Access (NUMA) fashion. Implementation results in a 28 nm FD-SOI technology show that 8 to 16 CPUs per cluster can be realized efficiently. In this work we focus on CPU clusters with an AXI interconnect and introduce a shared L1 data memory that can be accessed by all CPUs with low latency. Several memory banks are connected via a dedicated memory interconnect to allow for concurrent access from multiple CPUs.

The main contribution of this work is a comparison of L1 memory architectures within CPU clusters: 1) cluster level shared memory, 2) local memory attached to each CPU and 3) a hybrid architecture that integrates both. To couple the CPUs and the shared L1 memory, we compare a traditional crossbar and a Mesh-of-Tree (MoT) topology. We present synthesis and place and route (P&R) results using 28 nm FD-SOI standard cell technology. In addition, we perform an analysis with stream-based applications to determine the appropriate number of shared memory banks for such applications.

II. Related Work

This section presents architectures of shared L1 memories from industry and academia. Most implementations feature a banking factor of 2, i.e. there are twice as many memory banks as there are CPU cores in each cluster.

978-1-4799-8670-5/15 $31.00 © 2015 IEEE
DOI 10.1109/MCSoC.2015.25

Rahimi et al. [7] present a synthesizable logarithmic interconnect to connect CPU cores with multi-banked L1 data memory. Different CPU cluster configurations and target frequencies are compared in a 65 nm technology. A CPU read request has a delay of a single clock cycle. This is achieved by using a shifted clock for the memories which complicates clock management and timing closure of the overall system. The CoreVA CPU used in this work features a memory read delay of 2 clock cycles, so we can omit a shifted memory clock in this work. Besides synthesis results, Rahimi et al. present layouts with up to 32 CPUs and 64 memory banks.

Kakoee et al. [8] extend the work of Rahimi et al. to be more reliable and variation-tolerant. This is achieved by integrating a controllable pipeline stage between the CPUs and memory banks. In [9] a shared L1 data cache is presented. Using the logarithmic interconnect network proposed by Rahimi et al., the best-case read latency is one clock cycle. The cache shows an area and power overhead compared to a shared L1 memory (5% to 30%), but allows for easier programming of the system. Our CoreVA-MPSoC features a unified programming model (cf. Section VI) and does not require a data cache [6].

Gautschi et al. [2] present a cluster with four OpenRISC CPUs and an 8-banked shared L1 memory. Using 28 nm FD-SOI technology, the cluster operates at a near-threshold voltage of 0.6 V. The architecture of the 3-stage pipeline OpenRISC is improved to increase the energy efficiency by 50%.

Dogan et al. [3] present a multiprocessor system for biomedical applications with eight cores, a shared L1 data memory with 16 banks and a shared instruction memory with 8 banks. The shared data memory can be configured to have banks that are accessed by a single CPU exclusively ("private banks"). Dogan et al. use a memory management unit (MMU) to segment private and shared memory banks. To allow for power gating of unused memory banks, the MMU can be configured to access active banks only.

The STM STHORM MPSoC [4] connects several CPU clusters via a Globally-Asynchronous Locally-Synchronous (GALS)-based NoC. Within a cluster, 16 CPUs can access a shared 256 kB 32-banked data memory via a logarithmic interconnect. This results in a banking factor of 2. The CPU cores have a private instruction cache. Each cluster integrates a 2-channel DMA controller that can access the shared L1 memory and the NoC interconnect. The MPSoC is programmable via OpenCL or a proprietary Native Programming Model. The authors do not present any detailed information about the considered logarithmic interconnect.

The Plurality HyperCore architecture [5] integrates 64 RISC-like CPU cores, a hardware scheduler, and shared L1 data and instruction caches. The CPUs and the two caches are connected via a "smart interconnect". No detailed information about the architecture of the interconnect is provided, but Plurality states that multicasting the same data to several CPUs is supported. The proposed instruction cache has a size of 128 kB and the data cache a size of 2 MB.

The NVIDIA Maxwell GPU family features several Streaming Multiprocessors (SM) with 128 shader ALUs ("CUDA cores"), each with a dedicated 32-banked shared data memory [10]. A shared memory bank is 32 bits wide and can broadcast data to multiple shader ALUs. In addition, a SM integrates shared L1 data- and instruction caches. The high-end GM204 GPU contains 16 SMs with a total number of 2048 shader ALUs and 2 MB L2 cache.

III. The CoreVA-MPSoC Architecture

The configurable VLIW processor architecture CoreVA is designed to provide high resource efficiency [11]. The CPU is realized as a typical Harvard architecture with separated L1 instruction and data memories (cf. Fig. 1). L1 data memory is optional if a shared L1 memory is available on cluster level (cf. Section IV). The 32 bit architecture features a six-stage pipeline. Besides arithmetic-logic-units (ALUs), there are dedicated units for multiply-accumulate (MAC) and division (DIV). Memory loads, multiplications and MAC operations have a latency of two cycles, all other instructions have a latency of one cycle. Both ALU and MAC-units support a 16 bit SIMD mode. The VLIW architecture is highly configurable at design-time, allowing the number of VLIW slots, their functional units (ALU, MUL, DIV) as well as the number of load/store units (LD/ST) to be configured. Thus it is possible to tailor the processor's performance to match the needs of a given application domain. In this work, we use a CPU configuration with 16 kB instruction-memory, 2 VLIW slots as well as 1 MUL, 1 DIV, and 1 LD/ST unit. Our C compiler tool chain based on LLVM supports VLIW and SIMD vectorization.

To verify our physical implementation flow, performance and power models, two chip prototypes based on the CoreVA CPU architecture have been manufactured in a 65 nm process. The CoreVA-VLIW ASIC contains 4 ALU, 2 MAC, 2 DIV

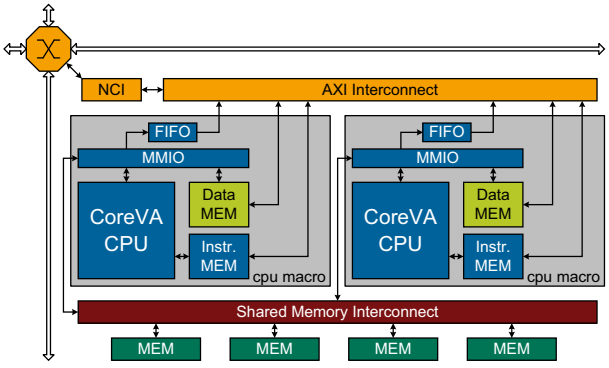


Figure 1. CPU cluster with Network-Cluster-Interface (NCI). The shared and the local L1 data memories are optional.

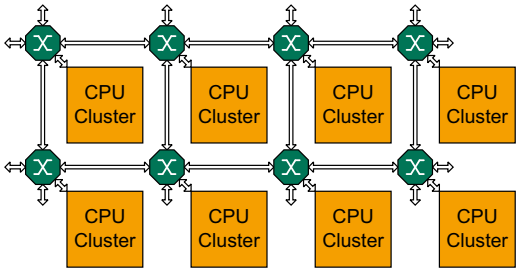


Figure 2. Hierarchical CoreVA-MPSoC with 4x2 mesh NoC.

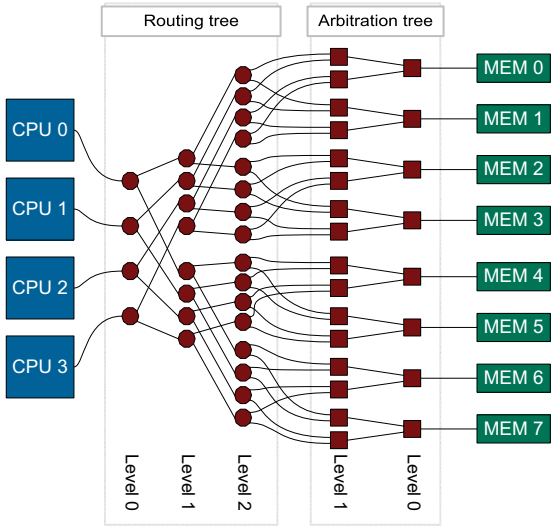
and 2 LD/ST units and is based on a conventional low power standard-cell library. This ASIC operates at up to 300 MHz and consumes about 100 mW at this frequency [11]. The RISC-like CoreVA-ULP [12] operates at voltages from 1.2 V down to 200 mV. A custom standard cell library optimized for sub-threshold operations is used. The CPUs lowest energy consumption per clock cycle (9.94 pJ) is observed at 325 mV.

A. CPU Cluster

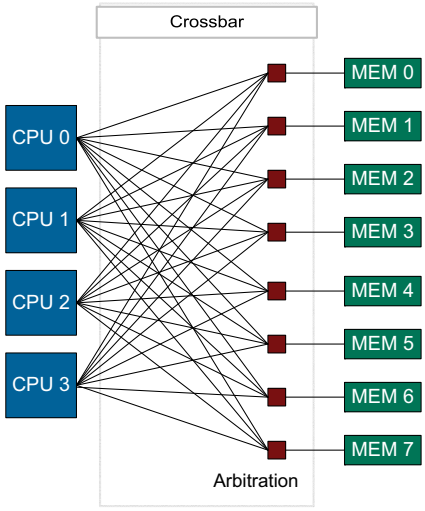
A CPU cluster tightly couples several CoreVA CPUs sharing a common address space. Within the basic configuration of a CPU cluster, each CPU can read and write the local L1 data memories of the other CPUs via a bus-based interconnect (cf. Fig. 1). Different bus standards, topologies (partial or full crossbar, shared bus), and data bus widths can be chosen at design time [6]. In this work, we use the AMBA AXI4 interconnect standard with a 32 bit bus width. AXI defines read and write channels and separate channels for address- and data transfers. This results in five channels in total: read and write data, read and write address, and write response). Register stages can be added to the interconnect to simplify P&R timing closure and to increase the maximum clock frequency of the MPSoC. The minimum CPU to CPU read latency without register stages is 4 clock cycles.

B. Network on Chip

To scale the CoreVA-MPSoC with dozens or hundreds of CPU cores, a Network on Chip (NoC) is used to connect several CPU clusters. The NoC features packet switching and wormhole routing. Packets are segmented into small flits each containing 64 bit payload data. Routing nodes, called switch boxes forward the flits along a path of network links. Each switch box has a configurable number of ports to allow for the implementation of different network topologies at design-time. Fig. 2 shows a 2D-mesh topology. Communication between two CPUs across the NoC is done via a network cluster interface (NCI, cf. Fig. 1). The NCI is connected to one port of each switch box and acts like a DMA controller by sending and receiving packets concurrently with CPU processing. The communication is done via uni-directional channels synchronized by mutexes. Packet data is directly stored to and read from each CPU's local data memory. To realize



(a) Mesh-of-Trees interconnect [7].



(b) Crossbar interconnect.

Figure 3. Interconnect topologies of shared L1 memory for 4 CPU cores and 8 memory banks.

large-scale CoreVA-MPSoCs we have extended our NoC by a Globally-Asynchronous Locally-Synchronous (GALS)-based approach [13]. To implement GALS, mesochronous links proposed in [14] are used between switch boxes to divide the system into smaller frequency domains.

IV. Tightly Coupled Shared L1 Memory Architecture

To further improve the resource efficiency and communication bandwidth of a CPU cluster, we integrate a multi-banked shared L1 data memory (cf. Fig. 1). Each CPU hard macro (cf. Section V) integrates an additional port (similar to the CPU's local memory interface) that connects the CPU's address decoder and the memory interconnect. If two CPUs claim access to the same memory bank, one

CPU has to be stalled for one clock cycle. To decrease the probability of such access conflicts, most shared L1 memory implementations have twice as many memory banks as CPUs [2]–[4]. This results in a banking factor of 2. In this work, we use the least significant address bits for bank selection to allow for fine-grained interleaving of accesses. In Section VI we investigate the appropriate number of memory banks for certain applications. The memory read latency of our CoreVA CPU for local data memory is two cycles. The shared L1 memory needs to have the same latency to omit additional pipeline stages or stall cycles. Thus, a fully combinational interconnect is required which results in tight timing requirements for the interconnect to allow for high clock frequencies. Within the first cycle of a memory access, the CPU calculates the memory address within the execute pipeline stage and passes the request to its address decoder and to the memory interconnect. The request is routed through the interconnect to the corresponding memory bank. At the next clock edge, the selected memory bank samples the request. In case of a read request, the requested data is passed to the interconnect, routed to the requesting CPU and sampled by a CPU-internal pipeline register.

For the memory interconnect we compare a Mesh-of-Trees (MoT) proposed in [7] with a crossbar implementation. The number of CPU and memory ports of both interconnects are configurable at design time. The MoT interconnect (cf. Fig. 3a) consists of a routing and an arbitration tree. The routing tree features routing nodes that transfer a memory request to the appropriate memory bank. The arbitration tree contains arbitration nodes that handle incoming requests from different CPUs. If there are two incoming requests at the same time, a round-robin arbitration is used. In case of a read request, each arbitration node stores the ID of the requesting CPU. In addition, each routing node stores the ID of the selected memory bank to allow for the routing of the data read from the bank back to the requesting CPU core. The crossbar interconnect (cf. Fig. 3b) connects every CPU to every memory bank in a point-to-point fashion. The interconnect integrates one non-pipelined round-robin arbiter per memory bank that handles requests from all CPUs. To route data from a memory bank back to the requesting CPU, the crossbar stores the IDs of the requesting CPUs in registers.

V. Implementation Results

This section compares physical implementation results for several CPU cluster configurations comprising different memory interconnect topologies and configurations. In comparison with performance results presented in Section VI, this allows us to determine the most resource efficient shared memory architecture for our CoreVA-MPSoC. We use a highly automated standard-cell design-flow based on Cadence Encounter Digital Implementation System. Basic blocks of our analysis are hard macros of our CoreVA CPU in a 2 VLIW slot configuration, 16 kB instruction memory, and

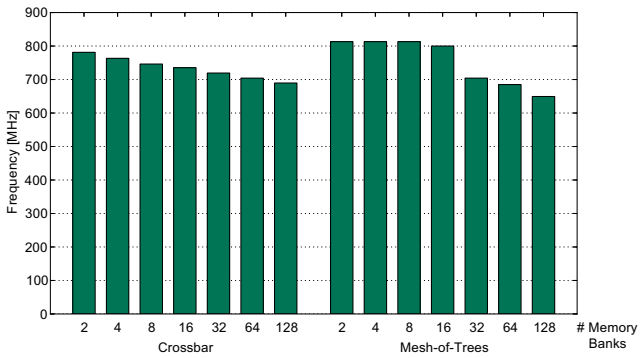


Figure 4. Maximum clock frequencies for different memory interconnects, number of memory banks, and 16 kB memory per bank.

32 kB, 16 kB, or 0 kB local data memory. The maximum frequency of the CoreVA CPU is 830 MHz in a 28 nm FD-SOI standard cell technology[1] with low- and regular-VT standard cells. Area requirements are 0.157 mm^2 (32 kB local data memory), 0.133 mm^2 (16 kB), or 0.096 mm^2 (0 kB). The 16 kB and the 0 kB macro feature a port to the shared L1 memory. We use an AXI bus as cluster interconnect for access to the local data and instruction memory of each CPU. The bus integrates two register stages so the bus does not limit the maximum clock frequency of our design.

Fig. 4 shows the maximum clock frequency of different shared L1 memory configurations. The CPU cluster integrates 8 CPU cores without local data memories. Each shared memory bank has a size of 16 kB and the number of memory banks varies from 2 to 128. For the MoT interconnect with 2 to 8 banks, the clock frequency is not limited by the shared memory subsystem but by the CPU. The critical paths of all other considered configurations traverse the memory interconnect. In detail, the path goes from one CPU through the memory interconnect and the arbiter back to another CPU. The memory macro is not part of this critical path so we can use area efficient high density memory blocks instead of high speed blocks. The MoT configuration with 16 banks has a slightly reduced frequency of 800 MHz whereas the MoT configurations with 32 to 128 banks achieve only 704 MHz to 649 MHz. Thus the configuration with 32 banks is 96 MHz slower than the 16 banks configuration. This high decrease can be explained by different optimization strategies of the synthesis tool. The crossbar interconnect shows a linear decrease of the clock frequency from 781 MHz (2 banks) to 689 MHz (128 banks). For 2 to 16 banks, the MoT interconnect shows an advantage compared to crossbar, but for more than 16 banks the crossbar is faster. Cluster configurations with 2, 4, and 16 CPUs benefit from the crossbar interconnect for higher bank counts as well.

Fig. 5 shows the area requirements for a CPU cluster with 8 CPUs, crossbar interconnect and different memory

[1] STMicroelectronics, 10 metal layer, Worst Case Corner: 1.0 V, 125°C

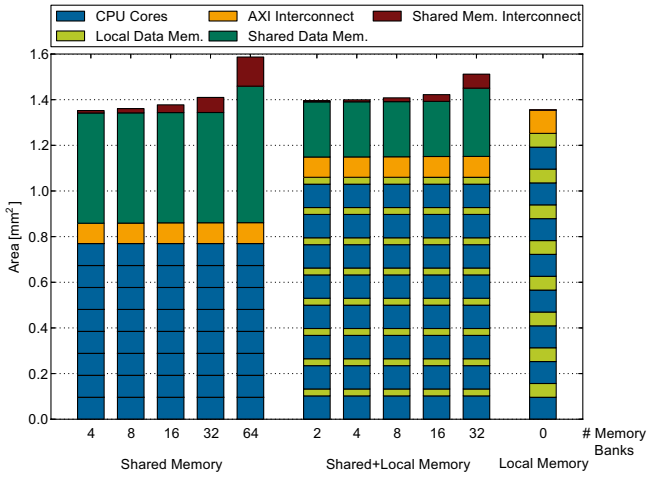
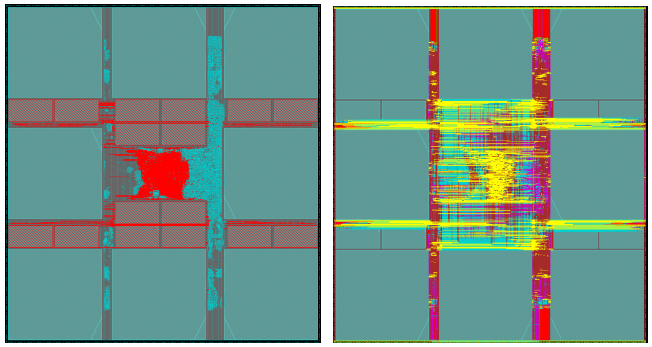


Figure 5. Area requirements of different CPU cluster configurations with crossbar interconnect, 256 kB data memory in total, and 650 MHz.



(a) Placed design (shared memory and its interconnect is marked red).

(b) Routed design.

Figure 6. Physical implementation of a CPU cluster with 128 kB shared L1 data memory, MoT interconnect, and 8 CPU macros that include 128 kB local data memory. Area requirements are 1.77 mm^2

configurations. Analysis shows that crossbar and MoT implementations of the memory interconnect do not differ significantly in area requirements. Total size of all data memories (shared and local) is fixed to 256 kB and a clock frequency of 650 MHz is used.

The *Shared Memory* configurations use the CPU macro with 0 kB data memory and 256 kB shared L1 data memory with 4 to 64 banks. In this configuration the CPU cores require an area of 0.77 mm^2. The shared L1 data memories with 4 to 32 memory banks are built from 8 kB memory blocks and require 0.483 mm^2. With 64 banks the shared data memory uses 4 kB memory blocks which increases the area by 23.8% to 0.598 mm^2. The area for the memory interconnect varies from 0.011 mm^2 (4 banks) to 0.128 mm^2 (64 banks). Considering the total area, the configuration with 64 memory banks is 17% larger compared to the one with 4 memory banks.

Shared+Local Memory configurations use the 16 kB CPU macro and 128 kB shared L1 memory. The number of memory banks is varied from 2 to 32. With 128 kB local data memory in total, the CPUs require an area of 1.06 mm^2. The shared L1 memory blocks require 0.241 mm^2 (4 to 16 banks) and 0.299 mm^2 (32 banks). The memory interconnect is slightly smaller compared to the *Shared Memory* configurations because it has to connect memory banks of half the size.

The bar *Local Memory* shows the configuration without shared L1 data memory and 32 kB local data memory per CPU. This configuration features a full AXI crossbar (0.102 mm^2) to allow for the concurrent communication of multiple CPUs. All other configurations use a shared AXI bus (ca. 0.090 mm^2) that is used for initialization and control of the CPUs. The minor difference is due to the two register stages that are integrated in both shared bus and crossbar. Nevertheless, the area overhead of an AXI crossbar is significant for larger CPU counts [6].

The three configurations *Shared Memory*, *Shared+Local Memory*, and *Local Memory* have comparable area requirements if the shared L1 memory features a small number of banks. Large number of memory banks (32 to 64) have an area overhead of up to 17% (*Shared Memory*) and 12% (*Shared+Local Memory*).

Fig. 6 presents the P&R results of a design with 8 CPUs, 16 kB local data memory per CPU (128 kB in total) and 128 kB shared L1 data memory with 16 banks (8 kB each) connected via the MoT interconnect. The CPU macros are placed in a three by three matrix with free space in the center for memory blocks and standard cells. Fig. 6a shows the placed design with the memory interconnect and the shared memory macros highlighted in red. The routed design indicates a routing hot spot in the center (cf. Fig. 6b). After P&R the area requirements (1.77 mm^2) increase by 16.8% compared to synthesis results. The maximum clock frequency is slightly reduced to 728 MHz compared to 800 MHz synthesis estimation (-9.9%). A design with the crossbar memory interconnect achieves the same maximum clock frequency and is 1% larger compared to the MoT layout. Post P&R power estimations results show a total power consumption of 231 mW including 17 mW per CPU. The memory interconnect logic consumes 18.4 mW with 28% cell-internal, 36% switching, and 36% leakage power.

VI. Benchmark Results

This section presents the impact of different partitioning and bank counts on the performance of several applications. In this work we use streaming applications written in the StreamIt language [15]. To map these applications consisting of a collection of filters to different MPSoC configurations, the compiler presented in [16] is used. Each filter reads an input data stream, processes the data and writes an output data stream. The filters communicate only via these unidirectional input and output channels and can be executed on different

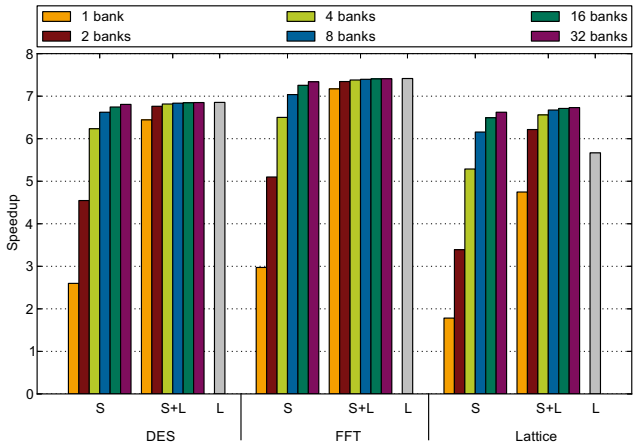


Figure 7. Speedup of *Shared* (S), *Shared+Local* (S+L), and *Local Memory* (L, grey bars) configurations, 1 to 32 banks, and 8 CPUs compared to a single CPU.

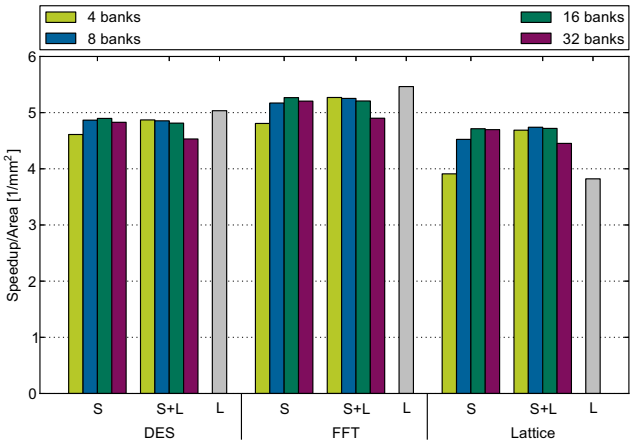


Figure 8. Speedup-area ratio of *Shared* (S), *Shared+Local* (S+L), and *Local Memory* (L, grey bars) configurations, 4 to 32 banks, 256 kB data memory in total and 8 CPUs.

CPUs in parallel. The mapping of filters to CPUs is done by a simulated annealing optimization algorithm. Our compiler for the StreamIt language generates C code that is compiled using an adapted LLVM compiler infrastructure and is executed on our cycle-accurate instruction set simulator. The performance of the applications is measured by a hardware clock counter attached to each CPU.

Communication between two CPUs within the same cluster or among different clusters is handled by a unified software communication library. This library implements communication channels handling buffer management and featuring a mutex-based synchronization scheme. Each channel consists of two or more buffers. This multi-buffering allows for hiding latencies by filling one buffer while another is being read. To avoid bus read latencies, the data buffers of a cluster-internal channel are allocated in the memory of the receiving CPU.

In this work, we focus on a single CPU cluster and the corresponding cluster communication channel. The local L1 data memory of each CPU guarantees a latency of two clock cycles for a read access. In contrast, reading from the memory of another CPU takes at least four cycles (cf. Section III). To achieve efficient communication with our local L1 memory architecture, our communication model relies solely on write accesses between CPUs [16]. The shared L1 memory architecture has a uniform two-cycle access latency from CPUs within the cluster.

The configurations *Shared Memory*, *Shared+Local Memory*, and *Local Memory* are compared with the same software partitioning. This partitioning is generated using the compiler for the StreamIt language and results in identical memory access patterns. The *Local Memory* configuration does not integrate a shared L1 memory. The CPUs communicate via the AXI interconnect. For *Shared+Local Memory*, the heap and the stack are located in the local memory whereas the communication buffers are in the shared L1 memory. The *Shared Memory* configuration allocates heap, stack, and communication buffers in the shared L1 memory.

We consider the streaming applications DES, FFT, and Lattice [15]. DES is an encryption algorithm whereas Lattice is from the signal processing domain. 30% of all instructions used by the DES application are load/store (LD/ST) operations. In detail, 11% are stores and 19% are loads. 24% of all LD/ST operations are used for CPU to CPU communication. 47% of all Lattice instructions are memory operations (27% load, 20% store). The ratio of LD/ST instructions for CPU to CPU communication is 34%. For FFT, 13% of all instructions are store operations and 23% are load operations. 23% of all LD/ST operations are used for communication.

Fig. 7 presents the speedup of the three StreamIt applications for cluster configurations with 1 to 32 memory banks and 8 CPUs compared to a single CPU with solely local memory. DES and FFT show the highest speedup (6.85 and 7.42) for the *Shared+Local Memory* configuration with 16 and 32 banks and the *Local Memory* configuration without shared L1 memory. Lattice shows the highest speedup of 6.73 for the *Shared+Local Memory* configuration with 32 banks. A *Shared+Local Memory* configuration with 256 banks (not shown on the plot) has a speedup of 6.75. This indicates that more than 16 memory banks are not required for the *Shared+Local Memory* configuration. The *Local Memory* configuration achieves only a speedup of 5.67. This is due to a high contention of the AXI interconnect. The number of bank conflicts is reduced by increasing the number of memory banks. The more memory banks are integrated in the shared L1 memory, the higher the speedup is. The *Shared Memory* configuration with Lattice and 1 Bank shows an average latency of 6.11 cycles due to memory bank conflicts. This results in a speedup of only 1.8. For 2 and 4 banks,

the average latency is 3.83 and 2.65 cycles respectively. A configuration with 32 banks shows an average latency of 2.05 which allows for a speedup of 6.6.

In general, the *Shared Memory* is outperformed by the *Shared+Local Memory* configuration with the same number of memory banks. This is because the shared L1 data memory is used for CPU to CPU communication only in case of *Shared+Local Memory*. The local data memory of each CPU contains its heap and stack and relieves the shared memory from up to 77% of all LD/ST operations.

For comparison of application performance and area requirements, we use the ratio of speedup and area as a metric. Fig. 8 shows the speedup-area ratio for cluster configurations with 4 to 32 memory banks. The *Local Memory* configuration shows the best speedup-area ratio for DES (5.03) and FFT (5.46), because the programming model considered in this work is tailored for this configuration. For Lattice the *Shared+Local Memory* with 16 memory banks shows the best speedup-area ratio of 4.74.

VII. Conclusion

This works presents memory topologies for tightly coupled L1 data memory within a CPU cluster of our embedded CoreVA-MPSoC. A CPU cluster can integrate local L1 data memories, a shared L1 memory, or a hybrid architecture that integrates both. We compare shared memory interconnects with crossbar and Mesh-of-Tree (MoT) topologies. Physical implementation results using 28 nm FD-SOI standard cell technology show a clock frequency advantage of the MoT interconnect for small memory bank counts (up to 16). In contrast the crossbar shows better results for higher bank counts. For a small number of memory banks (up to 16) the shared L1 data memory has an area overhead of only 2% compared to *Local Memory* of the same configuration (8 CPUs and 256 kB total memory per cluster). Runtime analysis with streaming applications shows better results for the *Shared+Local Memory* configuration with local and shared L1 data memory compared to the pure *Shared Memory* configuration. In combination with physical implementation results, 16 memory banks for the *Shared Memory* configurations and 4 banks for the *Shared+Local Memory* configuration allow for a high resource efficiency. Future work includes more advanced synchronization, programming models, and memory handling within a CPU cluster. In addition, we will connect the NoC to the shared L1 data memory.

Acknowledgments

This work was funded as part of the DFG Cluster of Excellence Cognitive Interaction Technology 'CITEC' (EXC 277), Bielefeld University and the BMBF Leading-Edge Cluster "Intelligent Technical Systems OstWestfalenLippe" (it's OWL), managed by the Project Management Agency Karlsruhe (PTKA). The authors are responsible for the contents of this publication.

References

[1] R. Banakar *et al.*, "Scratchpad Memory: Design Alternative for Cache On-Chip Memory in Embedded Systems," in *Int. Symp. on Hardware/Software Codesign (CODES)*. ACM Press, 2002, pp. 73–78.

[2] M. Gautschi *et al.*, "Customizing an Open Source Processor to Fit in an Ultra-Low Power Cluster with a Shared L1 Memory," in *Great Lakes Symp. on VLSI (GLSVLSI)*. ACM Press, 2014, pp. 87–88.

[3] A. Y. Dogan *et al.*, "Multi-Core Architecture Design for Ultra-Low-Power Wearable Health Monitoring Systems," in *DATE*. IEEE, 2012, pp. 988–993.

[4] L. Benini *et al.*, "P2012: Building an Ecosystem for a Scalable, Modular and High-Efficiency Embedded Computing Accelerator," in *DATE*. IEEE, 2012, pp. 983–987.

[5] J. Turley, "Plurality Gets Ambitious with 256 CPUs," *Microprocessor Report*, 2010.

[6] G. Sievers *et al.*, "Evaluation of Interconnect Fabrics for an Embedded MPSoC in 28nm FD-SOI," in *ISCAS*, 2015.

[7] A. Rahimi *et al.*, "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters," in *DATE*. IEEE, 2011.

[8] M. R. Kakoee *et al.*, "A Resilient Architecture for Low Latency Communication in Shared-L1 Processor Clusters," in *DATE*. IEEE, 2012, pp. 887–892.

[9] M. R. Kakoee *et al.*, "A Multi-Banked Shared-L1 Cache Architecture for Tightly Coupled Processor Clusters," in *Int. Symp. on System on Chip (SoC)*. IEEE, 2012.

[10] L. Case and D. Kanter, "Maxwell Illuminates the Masses," *Microprocessor Report*, 2014.

[11] B. Hübener *et al.*, "CoreVA: A Configurable Resource-Efficient VLIW Processor Architecture," in *Int. Conf. on Embedded and Ubiquitous Computing (EUC)*. IEEE, 2014, pp. 9–16.

[12] S. Lütkemeier *et al.*, "A 65 nm 32 b Subthreshold Processor With 9T Multi-Vt SRAM and Adaptive Supply Voltage Control," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 8–19, 2013.

[13] T. Jungeblut *et al.*, "A TCMS-based Architecture for GALS NoCs," in *ISCAS*. IEEE, 2012, pp. 2721–2724.

[14] D. Ludovici *et al.*, "Comparing tightly and loosely coupled mesochronous synchronizers in a noc switch architecture," in *Int. Symp. on Networks-on-Chip (NOCS)*. IEEE, 2009, pp. 244–249.

[15] W. Thies *et al.*, "StreamIt: A Language for Streaming Applications," in *Int. Conf. on Compiler Construction (CC)*. Springer, 2002, pp. 179–196.

[16] W. Kelly *et al.*, "A Communication Model and Partitioning Algorithm for Streaming Applications for an Embedded MPSoC," in *Int. Symp. on System on Chip (SoC)*. IEEE, 2014.