Scratchpad Memories with Ownership

Martin Schoeberl, Tórur Biskopstø Strøm, Oktay Baris, and Jens Sparsø

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: masca@dtu.dk, tbst@dtu.dk, okba@dtu.dk, jspa@dtu.dk

***Abstract*—A multicore processor for real-time systems needs a time-predictable way to communicate data between different threads running on different cores. Standard multicore processors support data sharing with shared main memory backed up by caches and cache coherence protocol. This sharing solution is hardly time predictable nor does it scale to more than a few cores.**

This paper presents a shared scratchpad memory (SPM) for time-predictable communication between cores. The base architecture uses time-division multiplexing for the arbitration of the access to the shared SPM. This allows the timing of programs executing on different cores to be completely independent of each other. We extend this architecture by the notion of ownership. A core can own the SPM. Having exclusive access to the SPM reduces the access time to a single clock cycle. The ownership of the SPM can then be transferred to a different core, implementing low latency communication of bulk data. As an extension, we propose to organize this memory as a pool of SPMs that can be owned by different cores and transferred as needed. We evaluate the proposed architecture within the T-CREST multicore architecture.

***Index Terms*—Scratchpad Memory, Multicore, Real-Time Systems**

I. Introduction

The gap between the access time of external memory and processor speed is getting larger and larger. This trend is further driven by multicore processors, where the individual cores share the main memory. For general purpose architectures, up to three levels of local and shared caches are used to "keep the bits on-chip". However, those levels of caches, and especially shared caches, are hardly predictable for real-time systems.

Core local scratchpad memories (SPM) are a viable solution for code and non-shared data. For shared data, a shared SPM, like a shared level 2 cache, can be used. However, in that case we need to consider the access time to this shared SPM with the worst case of all cores accessing this SPM. For time-predictable access to this shared SPM, we use time-division multiplexing (TDM) arbitration.

This paper presents a shared SPM with ownership. To avoid the access latency of a TDM arbitrated SPM we propose an SPM with ownership. A single core temporarily "owns" an SPM, writes data into it, and transfers the ownership to another core, the consumer of the data. We extend the idea from a single SPM to a pool of SPMs.

Using SPMs with a transfer of ownership is efficient and a time-predictable form of communicating bulk data between two or more cores. An example application is a digital signal processing pipeline where each pipeline stage acts as consumer and producer of sample data.

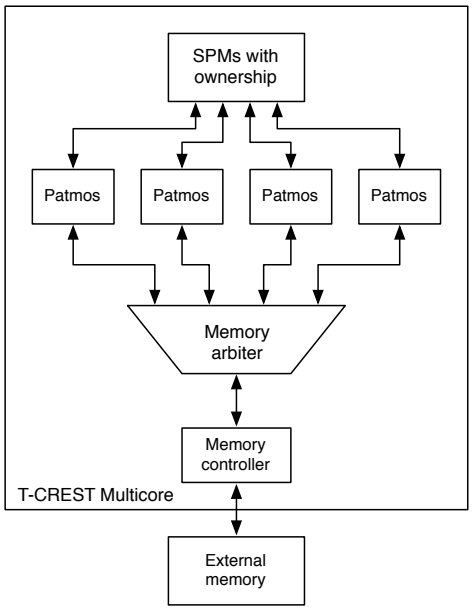

Fig. 1. The T-CREST multicore with a shared scratchpad memory with ownership.

If we want to avoid the transfer of ownership at runtime, but having an SPM that is shared within a subset of the cores, we can introduce this set of owners and provide a TDM based arbitration for exactly those cores sharing the SPM. The TDM schedule is set at application start and considered constant during runtime to enable a tight worst-case execution time (WCET) analysis.

To explore the SPM with ownership, we need a time-predictable multicore processor. T-CREST is such a multicore processor that is designed for hard-real time systems [13], [16]. As T-CREST is available in open source, we are able to extend this platform with the shared SPM with ownership for moving larger messages between cores. Figure 1 shows the T-CREST multicore, consisting of Patmos [17] processors, connected to the SPMs with ownership. To keep the figure comprehensible, we do not show the connections of all the cores to the Argo network-in-chip [7].

The contribution of this paper is a time-predictable and efficient solution to support bulk data communication between

cores on a multicore processor.

This paper is organized in 6 sections: The following section presents related work. Section III presents the concept of the SPM with ownership. Section IV describes how the SPMs with ownership can be used in a processing pipeline. Section V evaluates the design with respect to hardware resource consumption and performance when used to communicate between processing cores. Section VI concludes the paper.

II. Related Work

The use of SPMs was originally proposed as an alternative to caches [4]. The simpler implementation may result in a smaller, faster, and lower power designs, and the explicit control of when data is transferred between a SPM and a lower level memory can be exploited to reduce execution time as well as worst-case execution time (WCET) in real time systems. Here is a large space for optimizations [2], [1], [21], [19], [5] and in multicore processors additional dimensions opens [20], [18].

A comparison between locked cache blocks and a scratchpad memory with respect to the WCET can be found in [12]. While former approaches rely on the compiler to allocated the data or instructions in the scratchpad memory an algorithm for runtime allocation is proposed in [10].

Whitham and Audsley propose a special form of SPM, which includes a scratchpad memory management unit (SMMU) [22], [24]. The SMMU is in charge to redirect pointers into the SPM when data is moved into the SPM. Therefore, the pointer aliasing issue is solved in hardware. With an enhancement of the SMMU to support read-only objects and tiling of larger data structures it is shown that the WCET with a SMMU/SPM is close to the average case execution time with a conventional data cache [23].

Similar use of SPMs is addressed in [14], [9] that presents shared (scratchpad) memories in a multicore with a statically scheduled, TDM arbitration. In these designs the memory access time seen by a processor grows linearly with the schedule period, i.e., the number of processors. In this paper, we propose having a pool of SPMs each shared by a (small) subset of processors that can be changed dynamically and used to communicate data among processors.

Our work has a different aim; to use a pool of SPMs for inter-core communication in a multicore processor intended for real-time applications. Closer to this perspective is the SPM in the processor clusters in the Kalray manycore processor [6]. The processor is organized in 16 clusters of 16 cores. The clusters are connected by a NoC with rate control at the sender to support time-predictable message passing. Each core within a cluster and the NoC are connected to a SPM with 16 independent memory banks. By carefully selecting allocation of data and access to the memory banks, the notion of ownership can be implemented in software for the Kalray cluster.

III. Scratchpad Memories with Ownership

An SPM is an on-chip memory that is mapped into a processor's address space. We propose using SPMs shared between processor cores for communication of bulk data between processor cores.

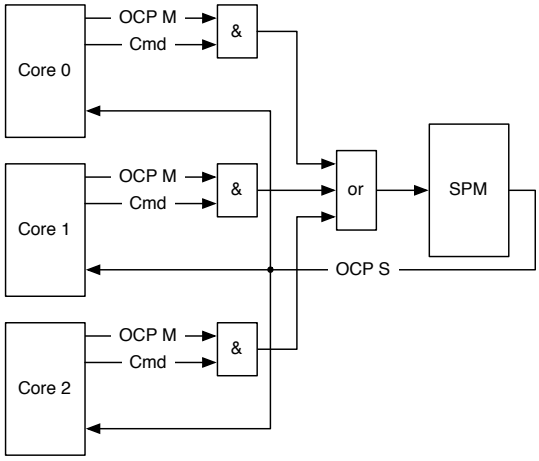

Fig. 2. A single SPM with ownership

A. A Single Shared SPM

The baseline of our design is an SPM that is shared between the cores. To provide time-predictable access to the SPM the arbitration is TDM where each core has a time slot for one memory access operation (read or write). Therefore, the worst-case latency for a memory access is to just miss the time slot and wait for all other cores' time slot. For the arbitration circuit, we register the request from a core, which results in an additional cycle latency. For n cores the WCET of a memory operation in clock cycles is $t_{acc} = n + 1$.

TDM based arbitration has two interesting properties: (1) it is time-predictable and (2) when all cores have a common notion of time, the arbitration can be distributed. The distributed arbitration results in a better scalability with the number of cores instead of a central arbitration scheme (e.g., round robin, or some form of priority).

B. Multiple SPMs with Ownership

With the notion of an ownership a core can exclusively own an SPM. With ownership we can reduce the access latency to a single clock cycle, which is the same access latency as for a core local SPM to $t_{acc} = 1$.

When only a single core can access the SPM at any point in time, the implementation can be simplified, compared to the shared SPM with TDM arbitration. Figure 2 shows a single SPM connected to several cores. Within T-CREST we use the OCP bus standard [11] for the connection of device and memories to the cores. A core is an OCP master with *OCP M* signals to a slave. The slave response is represented by the *OCP S* signal. Each core uses its own *Cmd* signal to enable the master signals (*OCP M*) with an AND gate. When the SPM is owned, only one core will issue a write or read command. Therefore, we can simply merge all master signals with an OR gate. The slave response (*OCP S*) is connected to all cores, as the value is ignored without an outstanding OCP command. The implementation of a single SPM with ownership is actually as simple as shown in Figure 2. Supporting a pool of SPMs is a little bit more evolved.

The ownership is not defined by the SPM itself. It is therefore necessary to manage the ownership in software. In our tests we use the shared SPM for the ownership management.

To allow concurrent communication between several cores, we provide several SPMs. A core can request one or more SPMs, use these SPMs, and later release the SPMs or transfer the ownership to another core.

The extension from a single SPM with a single owner to multiple single owned SPMs is implemented via address mapping. Each SPM is mapped to a unique address and address decoding selects the SPM that a core owns. The assignment of an SPM, and therefore a start address, represents the ownership.

C. Multiple SPMs with Multiple Owners

We can extend the idea of a single (exclusive) owner to a scenario where *several* cores have ownership of an SPM. This resembles a conventional shared memory, but with one main difference: that sharing is limited to a (typically small) subset of the cores. We use TDM arbitration to provide time-predictable access to shared memories. The length of the TDM schedule is the number of cores that currently share/own a SPM. With multiple SPMs, it is possible to arrange that different subsets of cores share different SPMs. These two aspects mean that a pool of SPM's each allowing multiple owners may offer lower access time and better scalability than a single shared memory. A core may have ownership to several SPMs. The concept includes exclusive ownership (as discussed in the previous subsection) as a special case.

The WCET for a memory access depends on the number of owners o and is $t_{acc} = o$.

To support the idea of multiple owners, it is necessary to introduce a mechanism that keeps track of which cores have ownership over which SPMs, which in turn defines each SPM's TDM-schedule. We implement this mechanism in hardware by giving each SPM a register that defines which cores are part of that SPM's TDM schedule. A hardware control unit is added that can read and write these registers. The control unit supports the following functions: (1) A core can request an SPM from the control unit. The control unit finds an available SPM, sets its schedule to contain the requested core, and returns its identification. If no SPM is available a negative result is returned. (2) Cores can access the acquired SPM by using the returned identification as an address offset into the SPM Pool. (3) Cores can set the TDM schedule of an SPM by writing it to the control unit. To return an SPM to the pool, cores simply set the SPM's schedule as empty.

IV. Communication with the SPMs

In this section, we outline how the different SPM configurations, introduced in the previous section, can be used to implement message passing inter-task communication. For simplicity, and without lack of generality, we assume that every task is assigned its own core. The tasks communicate using message passing and the message buffers are allocated in an SPM. Figure 3(a) shows a pipelined connection of tasks, mapped to different cores. Below we discuss how a task-to-task communication channel is implemented. The same techniques can be used in more complex task graphs where tasks have more inputs (that join) and/or more outputs (that fork).

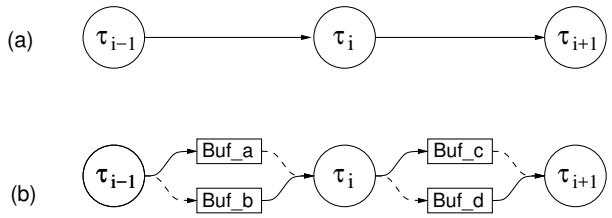

Fig. 3. (a) A chain/pipeline of communicating tasks. (b) Each communication channel is implemented using a pair of buffers that are mapped to one or more SPMs.

A task repeatedly: (i) reads inputs, (ii) computes, and (iii) writes outputs. To avoid copying of data, we assume that a task has exclusive access to its input and output buffers while computing. To allow concurrent operation of tasks, a double-buffered implementation is needed where a pair of buffers are used in an alternating fashion as illustrated in Figure 3(b). Task τ_i receives input data from task τ_{i-1} and it produces outputs from task τ_{i+1}. As illustrated in Figure 3(b), the double buffering means that task τ_i either reads from Buf_b and writes to Buf_c or reads from Buf_a and writes to Buf_d.

To ensure consistency of the data being communicated, a reader or a writer must have exclusive access to the relevant buffer. This requires at least one binary flag for each buffer marking whether the buffer is full or empty: A producer will wait for the relevant buffer to become empty, fill it and finally set the status to full. A consumer will wait for the relevant buffer to become full, read the message and set finally the status to empty. The producer and consumer must both have access to the flag.

In the evaluation that follows, we assume that all messages are the same size, n_w words, and we model tasks with real code for accessing the buffers.

V. Evaluation

We explore three different hardware configurations: (1) the shared SPM with TDM based arbitration, (2) a pool of SPMs with single ownership, and (3) a pool of SPMs with multiple owners that are arbitrated with TDM.

A. Evaluation Setup

We have implemented all three version of the shared SPM in the hardware construction language Chisel [3]. We integrated our SPMs with ownership into the T-CREST processor. By using Chisel as hardware construction language we can generate Verilog code for synthesis in an FPGA (or ASIC) or a cycle accurate emulation form the same hardware description.

We evaluate our design using the Altera DE2-115 development board. The FPGA on this board, the Intel/Altera Cyclone IV EP4CE115 FPGA, is big enough to build a system with up to 9 cores.

The Patmos cores are configured with a single-issue pipeline, an 8 KB method cache with 16 methods, a 4 KB write-through

TABLE I
RESOURCE CONSUMPTIONS AND MAXIMUM CLOCK FREQUENCY.

SPM	Cores	SPMs	LEs	DFFs	Fmax
Shared	8	1	646	435	72.2 MHz
Ownership	8	16	5465	68	60.7 MHz
Multi-Owner	8	16	6713	690	49.4 MHz
Patmos core	1	2	9437	4384	83.1 MHz
Argo 3x3 NoC	9	9	15146	8342	71.6 MHz

TABLE II
ACCESS TIME TO THE SPM WITH DIFFERENT CONFIGURATIONS.

Configuration	# Cores	# SPMs	Access time
Shared SPM	8	1	2–9
SPMs with ownership	8	16	1
Multi-Owner with 1 owner	8	16	1
Multi-Owner with 2 owners	8	16	1–2
Multi-Owner with 4 owners	8	16	1–4
Multi-Owner with 8 owners	8	16	1–8

data cache, a 2 KB stack cache, a 1 KB instruction SPM and a 2 KB local SPM. External memory is 2 MB with an access time of 21 clock cycles for a burst of 4 32-bit words for a single core. For multicores the main memory is TDM arbitrated, resulting in an access time between 21 and $n \times 21$ clock cycles for n cores. The SPMs are configured such that the number of SPMs multiplied with each SPM's size equals 16 KB, i.e., the single shared SPM is 16 KB whereas the 16 SPMs with ownership are 1 KB each.

B. Resource Consumption and Maximum Clock Frequency

Table I shows the resource consumptions of different configurations of the SPM with ownership. We obtain the synthesis results with the Quartus Prime Lite Edition. The resource consumption is given in logic elements (LE) containing a 4-bit lookup table (LUT) and D flip-flops (DFF). We report the maximum clocking frequency (Fmax) from the slow setting at 1200 mV and 85 C. The maximum clock frequency is for the whole multicore system.

To set the resource consumption in relation, the Table also shows how a single Patmos core consumes around 9000 LEs. Furthermore, we show the resource consumption of the Argo NoC, which is the default message passing support in T-CREST. Argo is a time-predictable NoC supporting message passing in hardware by using a DMA in the network interface. Therefore, the resource consumption is higher than for the SPM solutions. Furthermore, the Argo NoC is a distributed architecture where the bandwidth scales with the number of cores. The Argo NoC also scales better with the maximum clock frequency.

We can observe that the shared SPM with a TDM based arbiter has a low resource consumption. The multicore version of T-CREST results in a lower maximum clock frequency than for a single Patmos core. As the TDM arbitration is distributed to the cores (each core has its own TDM schedule counter) there are no scalability issues with 8 cores. Both the SPM with ownership and the multi-owner SPM have a longer critical path by allowing single cycle access, resulting in a lower maximum clock frequency. Te keep a high maximum clock frequency, we may introduce pipelining, at the cost of additional access latency, in the future.

We expect that none of the SPMs are feasible for manycore processors, with hundreds of cores. In that case, we envision shared SPMs for clusters of cores, similar to the Kalray design. The clusters sharing an SPM may overlap so that the border nodes of a cluster are members of two, or more, pools of shared SPMs.

C. Access Time

Table II shows the measured access times for a one word read or write for different SPM configurations. As a baseline, we measure access time to a plain shared SPM with TDM based arbitration. To avoid synchronizing with the TDM schedule, we precede each measurement with a random delay. Patmos contains a deadline device [15], which is similar to a deadline instruction [8]. With the deadline device we can stall the pipeline for a fixed number of clock cycles to generate a cycle accurate delay. We use this deadline device to generate the random delay.

For the access to the TDM arbitrated SPM, we observe all possible access times, i.e., for the 8 core version between 2 and 9 clock cycles. We perform the same measurement with the SPM with ownership. As expected we observe a constant access time of 1 clock cycle, independent of the number of cores or number of SPMs. The same measurement for the multi-owner SPM delivers different results depending on the number of owners configured for an SPM. For o owners we observe access times between 1 and o clock cycles. The measurements confirm the analytical WCET bounds for the access time.

D. Message Passing Communication

The intention of the SPM with ownership is to support communication of larger data structures between cores. In the shared SPM case this means writing the data by one core, transferring the data by sending a pointer to the data and reading the data on the destination core. For a SPM with ownership the transfer of the data is the transfer of the ownership of the SPM, e.g., by sending the SPM name, which is its address, to the consumer.

To allow maximum throughput, we use a double buffering scheme where the producer writes into one buffer while the consumer can read from the other buffer. Each buffer needs an associated flag to state if the buffer is empty or full, or in other words, is owned by the producer or consumer. This flags needs read and write access from both, the producer and the consumer. Therefore, it needs to be allocated in a memory area where access to both threads is allowed.

We evaluate a single producer thread and a single consumer thread executing on two cores. We transfer 16 KB of data in buffer sizes between 4 and 128 32-bit words. As we want

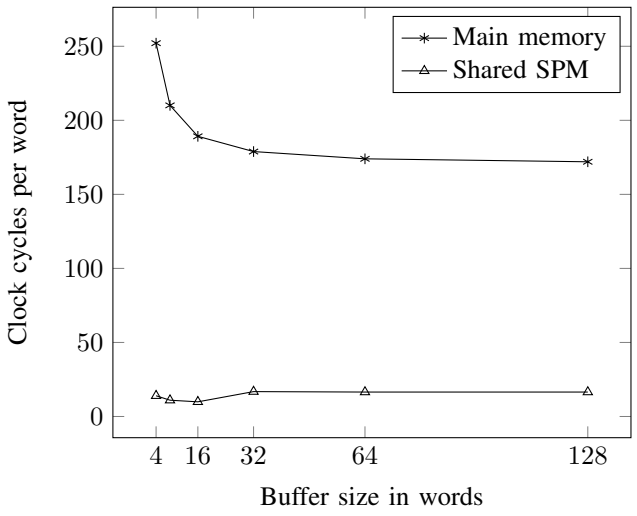

Fig. 4. Comparison of execution time in clock cycles per word transferred from a producer core to a consumer core for the main memory and the shared SPM.

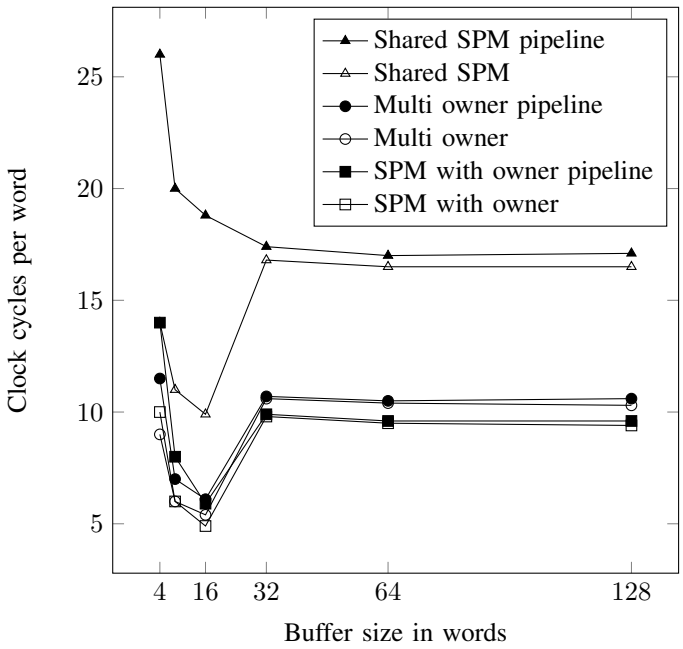

Fig. 5. Execution time in clock cycles per word transferred from a producer core, possibly through an intermediate/pipeline core, to a consumer core, for all SPMs.

to measure the communication efficiency, we use an artificial workload with low computation demands. The producer fills the buffer with a constant before transferring it to the consumer. That means the producer writes to each word in the buffer. The consumer reads each word from the buffer and adds the value to a sum, which is printed out at the end for testing purpose and to avoid that the compiler optimizes away the test code.

Furthermore, we also measure a pipeline design consisting of three nodes: (1) a producer filling the buffer with a constant, (2) a pipeline stage copying from the input buffer into the output buffer, and (3) a consumer reading the input buffer and summarizing the values.

Figure 4 shows the execution time in clock cycles per word transferred from a producer core to a consumer core when using main memory and when using the shared SPM. For both the main memory and the shared SPM, both buffers and flags are located in the same memory. The communication via shared main memory takes between 252 cycles for a 4-word buffer and 172 clock cycles for a 128-word buffer. In contrast, the communication with the shared SPM takes between 10 and 17 clock cycles per word. Interestingly there is a minimum at the 16 words buffer. This minimum result from unrolling the loop for up to 16 iterations. The external memory communication with around 170 clock cycles per word is dominated by the memory access time (a 21 clock cycle TDM slot and 8 cores is in the range of 170 clock cycles). The shared SPM with around 16 clock cycles per word, with a worst case access latency of 9 clock cycles, is partially computation bound and partially access time bound. Overall, we can observe that the shared on-chip SPM is by a factor of around 10 faster than using external main memory.

Figure 5 shows the execution time in clock cycles per word transferred from a producer core, possibly through an intermediate/pipeline core, to a consumer core for all SPMs. For the SPM with ownership, we allocate the handshake flags in a shared SPM. For all but the 4-word buffer, this solution gives the best performance. At 16 word buffer and loop unrolling the cost per word is 4.9 cycles. For larger buffer sizes, without loop unrolling, the cost per word is around 10 clock cycles.

The SPM with multiple owners uses two buffers with two owners for the double buffering. A buffer's flag is located after the buffer in the same SPM. Therefore, access to the buffers and the handshake flags is between 1 and 2 clock cycles. This results in a latency between 5.4 and 10.6 clock cycles. Here we also observe the effect of loop unrolling, where the sweet spot for communication is 16 words with a latency of 5.4 clock cycles per word. The SPM with multiple owners performs slightly better than the SPM with single ownership, when the buffers are small, as the access time for the flags has a larger effect.

For all SPMs, adding an intermediate/pipeline core adds just minimal overhead to the throughput. However, for the shared SPM with a buffer size less than 32 words, there is a similar effect as for the main memory, where the communication overhead is so big that it offsets the benefits of loop unrolling.

In summary, the shared SPM is the easiest solution to use and provides far better performance than main memory. However, combining the shared SPM with SPMs with ownership provides up to twice the performance of just the shared SPM. The multi-owner SPMs reduce the TDM schedule to only 2 cycles. However, this is not as advantageous as reading/writing data at 1 clock cycle and reading/writing flags at up to 9 clock cycles.

E. Source Access

We evaluated our SPM with ownership in the open-source multicore T-CREST. Therefore, we also provide our imple-

mentation in open source. Instructions on how to build T-CREST and how to reproduce the results can be found at https://github.com/t-crest/patmos/tree/master/c/apps/ownspm.

VI. Conclusion

Multicore processors used in real-time systems need a time-predictable and efficient way to communicate data between the cores. This paper presents a shared scratchpad memory (SPM) with ownership. We provide a pool of SPMs, where each SPM can be owned by one or more cores. The ownership can change at runtime. This allows to build efficient communication mechanism for bulk data: (1) one core writes data into such an SPM, (2) transfers the ownership to another core, which (3) consumes the data. This mechanism can be used for double buffering for a processing pipeline.

We have implemented three versions of the shared SPM and evaluated the shared SPMs with test programs emulating a processing pipeline. With buffers in the range of 128 bytes and more, the execution time is basically the inner loop to fill (or consume) the data, which is with the Patmos processor 10 clock cycles per 32-bit word.

Acknowledgment

The work presented in this paper was partially funded by the Danish Council for Independent Research | Technology and Production Sciences under the project PREDICT(no. 4184-00127A) and by the Faroese Research Council (no. 0607).

References

[1] Federico Angiolini, Luca Benini, and Alberto Caprara. Polynomial-time algorithm for on-chip scratchpad memory partitioning. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES-03)*, pages 318–326, New York, October 30 November 01 2003. ACM Press.

[2] Oren Avissar, Rajeev Barua, and Dave Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. *Trans. on Embedded Computing Sys.*, 1(1):6–26, 2002.

[3] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanovic. Chisel: constructing hardware in a scala embedded language. In *The 49th Annual Design Automation Conference (DAC 2012)*, pages 1216–1225, San Francisco, CA, USA, June 2012. ACM.

[4] Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, and Peter Marwedel. Scratchpad memory: Design alternative for cache on-chip memory in embedded systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, CODES '02, pages 73–78, New York, NY, USA, 2002. ACM.

[5] Jean-Francois Deverge and Isabelle Puaut. Wcet-directed dynamic scratchpad memory allocation of data. In *ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 179–190, Washington, DC, USA, 2007. IEEE Computer Society.

[6] Benoît Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. Time-critical computing on a single-chip massively parallel processor. In *Conference on Design, Automation and Test in Europe*, DATE '14, pages 97:1–97:6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.

[7] Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christian T. Müller, Kees Goossens, and Jens Sparsø. Argo: A real-time network-on-chip architecture with an efficient GALS implementation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24:479–492, 2016.

[8] Ben Lickly, Isaac Liu, Sungjun Kim, Hiren D. Patel, Stephen A. Edwards, and Edward A. Lee. Predictable programming on a precision timed architecture. In Erik R. Altman, editor, *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008)*, pages 137–146, Atlanta, GA, USA, October 2008. ACM.

[9] Emad Jacob Maroun, Henrik Enggaard Hansen, Andreas Toftegaard Kristensen, and Martin Schoeberl. Time-predictable synchronization support with a shared scratchpad memory. *Microprocessors and Microsystems*, 64:34 – 42, 2019.

[10] Ross McIlroy, Peter Dickman, and Joe Sventek. Efficient dynamic heap allocation of scratch-pad memory. In *ISMM '08: Proceedings of the 7th international symposium on Memory management*, pages 31–40, New York, NY, USA, 2008. ACM.

[11] OCP-IP Association. Open core protocol specification 2.1. http://www.ocpip.org/, 2005.

[12] Isabelle Puaut and Christophe Pais. Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE 2007)*, pages 1484–1489, San Jose, CA, USA, 2007. EDA Consortium.

[13] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[14] Martin Schoeberl, David VH Chong, Wolfgang Puffitsch, and Jens Sparsø. A time-predictable memory network-on-chip. In *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, pages 53–62, Madrid, Spain, July 2014.

[15] Martin Schoeberl, Hiren D. Patel, and Edward A. Lee. Fun with a deadline instruction. Technical Report UCB/EECS-2009-149, EECS Department, University of California, Berkeley, October 2009.

[16] Martin Schoeberl, Luca Pezzarossa, and Jens Sparsø. A multicore processor for time-critical applications. *IEEE Design Test*, 35:38–47, 2018.

[17] Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, Apr 2018.

[18] Majid Shoushtari, Bryan Donyanavard, Luis Angel D. Bathen, and Nikil Dutt. Shave-ice: Sharing distributed virtualized spms in many-core embedded systems. *ACM Trans. Embed. Comput. Syst.*, 17(2):47:1–47:25, February 2018.

[19] Vivy Suhendra, Tulika Mitra, Abhik Roychoudhury, and Ting Chen. WCET centric data allocation to scratchpad memory. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS)*, pages 223–232. IEEE Computer Society, 2005.

[20] Hossein Tajik, Bryan Donyanavard, Nikil Dutt, Janmartin Jahn, and Jörg Henkel. Spmpool: Runtime spm management for memory-intensive applications in embedded many-cores. *ACM Trans. Embed. Comput. Syst.*, 16(1):25:1–25:27, October 2016.

[21] Lars Wehmeyer and Peter Marwedel. Influence of memory hierarchies on predictability for time constrained embedded software. In *Proceedings of Design, Automation and Test in Europe (DATE2005).*, pages 600–605 Vol. 1, March 2005.

[22] Jack Whitham and Neil Audsley. Implementing time-predictable load and store operations. In *Proceedings of the International Conference on Embedded Software (EMSOFT 2009)*, 2009.

[23] Jack Whitham and Neil Audsley. Investigating average versus worst-case timing behavior of data caches and data scratchpads. In *Proceedings of the 2010 22nd Euromicro Conference on Real-Time Systems*, ECRTS '10, pages 165–174, Washington, DC, USA, 2010. IEEE Computer Society.

[24] Jack Whitham and Neil Audsley. Studying the applicability of the scratchpad memory management unit. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '10, pages 205–214, Washington, DC, USA, 2010. IEEE Computer Society.