Towards General Purpose Tagged Memory

Wei Song, Alex Bradbury, and Robert Mullins
Computer Laboratory, University of Cambridge, Cambridge CB3 0FD United Kingdom
{firstname.lastname}@cl.cam.ac.uk

I. Introduction

lowRISC is a not-for-profit project [1] aiming to produce high quality open source System-on-Chip (SoC) implementations that can be exploited by academia, industry, and the wider open-source community. We have previously described our plans for tagged memory and 'minion' cores [2] as key features of our hardware platform. These are both flexible system-level features with uses including security and software-based specialisation. One of the most interesting use-cases for tagged memory is in providing protection against control-flow hijacking attacks [2], [3]. We also intend to further explore a wide range of additional potential uses including fine-grained memory synchronisation, garbage collection, and debug tools. To further explore implementation costs and performance implications of tagged memory, we have extended the Rocket RISC-V implementation [4] with preliminary tagged memory support. The current implementation extends on-chip caches to hold tags, adds a tag cache, and adds minimal instruction set extensions to manipulate tags. A richer integration into the instruction set and the ability to configure the triggering of interrupts on access to memory which has been given a certain tag will be added in future development.

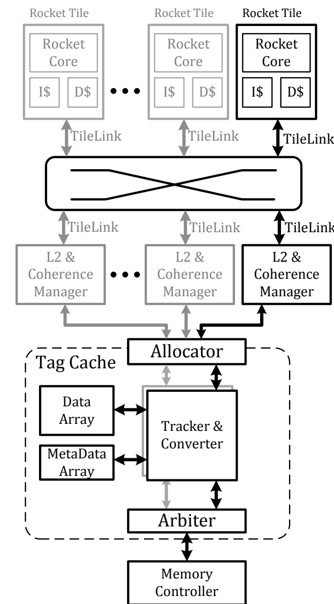

Fig. 1. Rocket chip with tag cache

II. Rocket Chip

The 'Rocket chip' is a configurable SoC generator which can instantiate a RISC-V Rocket core and associated memory hierarchy. The generated Rocket chip SoC comprises a group of Rocket tiles, each of which contains a RISC-V Rocket core running the 64-bit RISC-V instruction set [5], an L1 instruction cache and a non-blocking L1 data cache. Coherence is maintain by "coherence managers" in each L2 bank. Communication between the Rocket tiles and coherence managers takes place using the TileLink protocol [6]. The protocol defines a number of independent transaction channels, the prioritization of the channels and their format.

III. Tagged Memory

The implementation of tagged memory adds a design-time configurable number of tag bits to each 64-bit word in memory. These tag bits are copied along with the data word through the cache hierarchy, meaning each word in the L1 data and L2 cache lines are augmented with additional tag bits. In the same way, the payload of all TileLink messages and L2 caches are augmented, allowing the coherence of tags to be maintained by the existing cache coherence protocol. Two new instructions (LTAG, STAG) have been added for loading and storing tags. An exception is raised if the memory address operand to LTAG or STAG is not word-aligned.

We choose to store tag metadata in a reserved area of main memory. An alternative implementation approach would use the ECC bits available in some DRAMs to store tags. This would avoid the need for a tag cache, although such an approach may still benefit from using a region of physical memory to store tag metadata for multiple granularity tags. Repurposing the ECC bits in this way would of course prevent their use for their intended purpose. It also relies on the ability to modify the memory controller to add the necessary logic and of course the use of ECC DRAM. Without an additional cache, an extra access to the reserved tag area would be needed for each access to main memory. In order to minimise this memory traffic overhead, a tag cache is added to the Rocket chip as depicted in Fig. 1. It has a similar structure to the tag cache used in CHERI [7] but with parallel handlers. In the case of a miss in the tag cache, the 64-byte cache line containing the associated tag is read from memory using a parallel memory access to the reserved area. Extra handlers can be configured to serve multiple memory requests in parallel.

IV. Performance Analysis

The number of tag bits is fully parameterisable at design time. For these experiments we add two tag bits to each 64-bit word. The implementation has been tested both in simulation and on FPGA using the RISC-V test suite, tests specifically targeting tagged memory, and existing software ports such as the Linux kernel. Experiments are run using a single core with a 2-way 8KiB instruction cache and 4-way 16KiB data cache. The tag cache is 8-way set associative.

TABLE I. MISS RATES AND MEMORY TRAFFIC FOR THE SPECINT 2006 BENCHMARK SUITE

	I$ 8KiB (MPKI)	D$ 16KiB (MPKI)	L2 256KiB (MPKI)	Mem Traffic without Tag (TPKI)	Tag$ 16KiB (MPKI)	Traffic Ratio	Tag$ 32KiB (MPKI)	Traffic Ratio	Tag$ 64KiB (MPKI)	Traffic Ratio	Tag$ 128KiB (MPKI)	Traffic Ratio
perlbench	20	5	<1	2	<1	1.289	<1	1.089	<1	1.025	<1	1.011
bzip2	<1	14	10	16	10	1.941	7	1.688	3	1.281	<1	1.007
gcc	15	11	4	6	2	1.497	<1	1.240	<1	1.072	<1	1.023
mcf	<1	168	104	136	67	1.651	40	1.409	11	1.128	3	1.040
gobmk	24	8	3	6	1	1.368	<1	1.146	<1	1.073	<1	1.046
sjeng	11	5	1	3	1	1.673	<1	1.482	<1	1.383	<1	1.316
h264ref	1	3	2	3	<1	1.480	<1	1.265	<1	1.109	<1	1.028
omnetpp	40	5	<1	<1	<1	1.653	<1	1.415	<1	1.190	<1	1.042
astar	<1	21	5	9	4	1.750	2	1.471	<1	1.173	<1	1.009
average	12	27	14	20	10	1.589	6	1.356	2	1.159	<1	1.058

The SPECInt 2006 benchmark suite (test data set) has been used to investigate the increase in memory traffic due to tags. Currently 9 (out of 12) benchmark cases have been successfully compiled and run on FPGA (with the aid of Speckle [8]). Table I reveals the miss rates (misses per K instructions, MPKI) of each cache and the memory traffic of the original Rocket chip (transactions per K instructions, TPKI). We explore tag cache sizes of 32, 64 and 128KiB and report the increase in memory traffic (compared with an implementation without tags).

Without a tag cache we would require an additional access to the reserved tag area for each memory transactions, resulting in a memory traffic ratio of 2. The use of a tag cache is effective in reducing this ratio. The largest tag cache reduces the overhead in additional memory traffic to a few percent in most cases. The overhead for the `sjeng` benchmark appears high but in practice absolute memory traffic is low in this case.

There is significant scope for further optimisation of the tag cache. The current set of results represent a worse case where every single memory access requires a corresponding tag access. In reality, it is likely that there will be large regions where tags are not applied. Optimising for these cases through a multiple granularity tagging scheme will reduce memory traffic and pollution in the tag cache. We are in the process of exploring options for such a hierarchical scheme. These options include a completely hardware managed scheme with cache line granularity, a scheme where the operating system is responsible for marking regions as cleared, or a more general compression scheme such as run-length encoding. Another potential optimisation would be to have the tag cache check if the tag is actually changed on a write, or to track the dirty state of tags in the L1 and L2 separately to the dirty state of the data. Properly evaluating these choices and their trade-offs requires a range of realistic workloads which we are in the process of developing through our investigations into potential applications of tagged memory. Even the most straightforward of these approaches would help provide the property that "you don't pay for what you don't use", i.e. software that doesn't make use of tagged memory would see almost zero runtime overhead.

V. FUTURE WORK AND CONCLUSION

We believe tagged memory is a low-cost feature that provides a valuable foundation for a range of useful features in a general purpose SoC. We hope to further demonstrate this in future work. We see a number of directions as being of particular interest:

1) Exploring a range of uses for tagged memory and appropriate ISA support.
2) Further reducing the overhead of tagged memory through hierarchical tagging schemes.
3) Schemes for configuring the interpretation of tags, including the triggering of interrupts, communication with minions and potentially the propagation of tags through the ALU. Simple approaches include the use of a fixed-sized table defining rules and actions (e.g. indexed by instruction tag and opcode class). One could also imagine a range of approaches based on simplified versions of the PUMP [9].
4) Using minion cores in concert with tagged memory to implement more complex security policies and higher level functionality such as a full capability system [7]. This would require the ability for a minion to subscribe to events indicating access to memory with certain tags. Such communication would take place over dedicated communication links.

REFERENCES

[1] "lowRISC project." [Online]. Available: http://www.lowrisc.org

[2] A. Bradbury, G. Ferris, and R. Mullins, "Tagged memory and minion cores in the lowRISC SoC," Decemeber 2014. [Online]. Available: http://www.lowrisc.org/downloads/lowRISC-memo-2014-001.pdf

[3] L. Szekeres, M. Payer, T. Wei, and D. Song, "SoK: Eternal war in memory," in *IEEE Symposium on Security and Privacy*, May 2013, pp. 48–62.

[4] "Rocket chip generator." [Online]. Available: https://github.com/ucb-bar/rocket-chip

[5] A. Waterman, Y. Lee, D. Patterson, and K. Asanović, "The RISC-V instruction set manual — volume I: user-level ISA," CS Division, EECE Department, University of California, Berkeley, May 2014. [Online]. Available: http://riscv.org/riscv-spec-v2.0.pdf

[6] H. Cook, "Tilelink 0.3.1 specification," February. [Online]. Available: https://github.com/ucb-bar/uncore/blob/master/doc/TileLink0.3.1Specification.pdf

[7] J. D. Woodruff, "CHERI: A RISC capability machine for practical memory safety," Ph.D. dissertation, Computer Laboratory, University of Cambridge, 2014, section 5.5.2. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-858.pdf

[8] C. Celio, "A wrapper for the SPEC CPU2006 benchmark suite," April. [Online]. Available: https://github.com/ccelio/Speckle

[9] U. Dhawan and et al, "Architectural support for software-defined metadata processsing," in *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.