Artifact Report: Intel PMDK Transactions: Specification, Validation and Concurrency*

Azalea Raad[1], Ori Lahav[2], John Wickerson[1], Piotr Balcer[3], and Brijesh Dongol[4](✉)

[1] Imperial College London, London, UK
[2] Tel Aviv University, Tel Aviv, Israel
[3] Intel, Gdansk, Poland
[4] University of Surrey, Guildford, UK
b.dongol@surrey.ac.uk

Abstract. This report extends §6 of the main paper by providing further details of the mechanisation effort.

1 Modelling and Validating Correctness in FDR4

FDR4 [4] is a model checker for CSP [5] that has recently been used to verify linearisability [7], as well as opacity and durable opacity [3]. We similarly provide an FDR4 development, which allows proofs of refinement to be *automatically* checked up to certain bounds. This is in contrast to manual methods of proving correctness of concurrent objects [2,1], which require a significant amount of manual human input (though such manual proofs are unbounded). FDR4 uses a variety of underlying model checking paradigms and partial-order reduction techniques [4], depending on the structure of the files to be verified. FDR4 builds on FDR3, but the exact implementation details of FDR4 are not publicly available since it is a commercial product (available for free academic use).

The CSP files corresponding to this report may be downloaded from [8].

1.1 Modelling Details

One of the most challenging aspects of the FDR4 development is the modelling work itself. Our algorithms execute over a shared memory, but the CSP formalism is based on *communicating processes* with no notion of shared states. Thus, for each location we must explicitly define *handler* processes that communicate through *channels* to update and return the values of components (e.g. the addresses, read/write sets) of each model. Moreover, the implementations (TXPMDK,

* Raad is funded by a UKRI fellowship MR/V024299/1, EPSRC grant EP/X037029/1 and VeTSS. Lahav is supported by the Israel Science Foundation (grant 814/22) and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 851811). Wickerson is funded by EPSRC grant EP/R006865/1. Dongol is funded by EPSRC grants EP/Y036425/1, EP/X037142/1, EP/X015149/1, EP/V038915/1, EP/R025134/2 and VeTSS.

S. Weirich (Ed.): ESOP 2024, LNCS 14577, pp. 180–184, 2024.
https://doi.org/10.1007/978-3-031-57267-8_7

PMDK-NOREC and PMDK-TML), the specification (DDTMS) and underlying memory models (PSC and $PTSO_{syn}$) we consider are non-trivial, significantly increasing the challenge of the modelling effort. Although constructing the models is challenging, once the models have been developed, they can be combined in a modular fashion. We have taken advantage of this feature to combine our implementations with different memory models during development. The combination of PMDK-TML and TML/NOrec also takes advantage of this modularity.

This modularity also means that our models are reusable. One could use our models to check other developments, e.g. those that use TXPMDK to implement other failure-atomic data structures, or verify redesigns of TXPMDK over different memory models. Specifically, we use a top-level CSP process (which may comprise an interleaved composition of processes for each transaction) to model the most general client. Each transaction process begins a transaction, and then calls an unbounded number of reads, writes and allocations at non-deterministically chosen locations and with non-deterministically chosen values. An in-flight transaction process may also non-deterministically choose to terminate by calling commit instead of calling a read, write or allocation. Each operation call produces an externally visible invocation event, and when the operation terminates, an externally visible response is generated. Some operations may respond with an abort, in which case the transaction process itself terminates.

Additionally, there is an externally visible crash event that synchronises with all processes. At the level of the abstraction (i.e. DDTMS), this simply terminates all in-flight transactions, and resets the memory sequence (as detailed by the rule (X)). At the level of the implementation, all in-flight transactions are terminated and additionally, the store and persistency buffers are cleared. This means that when execution resumes, the value of each location is taken from NVM. Immediately after a crash (and before any other processes are started), the recovery process corresponding to the algorithm is executed. Note that transaction identifiers are never reused.

We eschew further details of our FDR4 models since they are provided as supplementary material [8] and also refer the interested reader to other prior works [7,3].

1.2 Overview of Development

An overview of our FDR4 development is given in Fig. 1. We derive two specifications from DDTMS. The first is an FDR4 model of DDTMS itself, based on prior work [7,3], but contains the extensions required for DDTMS. The second is DDTMS-SEQ, which restricts DDTMS to a sequential crash-free specification. We use DDTMS-SEQ to obtain (lower-bound) liveness-like guarantees, which strengthens traditional deadlock or divergence proofs of refinement. These lower-bound checks ensure our models contain at least the traces of DDTMS-SEQ.

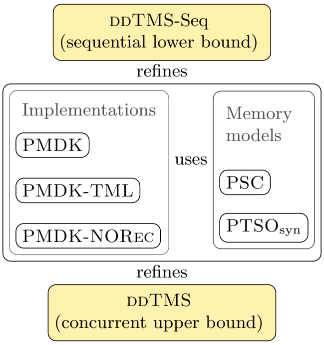

Fig. 1: Overview of FDR4 checks

Memory	#txns	#locs	#val	#buff	TXPMDK	PMDK-TML	PMDK-NOREC
PSC	2	2	2	2	5.83s	5.90s	6.74s
PSC	2	3	2	2	201.03s	213.97s	271.35s
PSC	2	2	3	2	21.65s	23.47s	27.40s
PSC	2	2	2	3	5.83s	5.78s	6.60s
PTSO$_{syn}$	2	1	2	2	0.61s	3.96s	1.57s
PTSO$_{syn}$	2	2	2	2	6.67s	6.71s	7.73s
PTSO$_{syn}$	2	3	2	2	267.1s	268.91s	319.18s
PTSO$_{syn}$	2	2	3	2	24.10s	25.53s	29.24s
PTSO$_{syn}$	2	2	2	3	14.37s	14.19s	15.41s

Fig. 2: Summary of upper bounds checks (total time in seconds: compilation + model exploration). The time out (TO) is set to 1000 seconds of compilation time.

CSP files. Our development comprises the following files.

File	**Description**
`Types.csp`	Contains the basic types and parameters. Use this file to increase / decrease the number of transactions, memory locations, values, etc. Defaults to 2 transactions, 2 locations and two values.
`MemoryP.csp`	Handler for memory, as well as the redo and undo logs. Operations query handlers to read/update the shared memory, flush to persistent memory and recover. This file is used to switch between memory models (NVM (which contains no crashes), PSC and PTSO$_{syn}$) - see the bottom of the file.
`LocHandler.csp`	Handler for local memory (i.e., the `loc` variable used by the implementations in Figs. 5 and 6.
`ddTMS.csp`	Model of the DDTMS automata from the main paper (Fig. 8).
`PMDK.csp`	Model of PMDK from Fig. 4 of the main paper.
`PMDK-TML.csp`	Model of PMDK-TML from Fig. 5 of the main paper.
`PMDK-NOrec.csp`	Model of PMDK-NOREC from Fig. 6 of the main paper.
`Refinement.csp`	File containing all checks to be performed.

Description of Tests. The file `Refinement.csp` comprises six tests as detailed in Figs. 9 and 10 of the paper. There are three upper-bound checks, which show that PMDK, PMDK-TML and PMDK-NOREC are refinements of DDTMS, validating soundness:

- `FinalTMS [T= PMDK`, checking that PMDK refines DDTMS.
- `FinalTMS [T= FinalTML`, checking that PMDK-TML refines DDTMS.
- `FinalTMS [T= FinalNOrec`, checking that PMDK-NOREC refines DDTMS.

Each of these tests can be run against the memory models: NVM (which contains no crashes), PSC and PTSO$_{syn}$ by commenting/uncommenting the relevant lines at the end of the file `MemoryP.csp`.

Additionally, there are three lower-bound checks, which show DDTMS-SEQ are refinements of PMDK, PMDK-TML and PMDK-NOREC.

- `PMDK [T= SeqFinalTMS`
- `FinalTML [T= SeqFinalTMS`
- `FinalNOrec [T= FinalNOrec`

Each of these tests can be run against the memory models: NVM and PSC as defined in the file `MemoryP.csp`. Note that the test against $\mathrm{PTSO}_{\mathrm{syn}}$ times out. However, the tests above are sufficient since $\mathrm{PTSO}_{\mathrm{syn}}$ reduces to PSC in the absence of data races (e.g., sequential executions).

Each check in FDR4 is split into two phases: **(1)** a compilation phase that builds the models; and **(2)** a model exploration phase. The characteristics of the upper and lower bounds checks are distinct. When naively checking the upper bound, compilation is almost instantaneous but model exploration times can be significant; these times are swapped for the lower bounds checks.

In general, lower-bounds take much longer to verify than the upper-bounds since FDR4 is optimised to verify abstract (low-detail) specifications are refined by concrete (high-detail) implementations. The lower bounds checks use the more complex models as the specification, leading to the creation of very large space-inefficient models, putting pressure on the available system memory. However, the lower-bound checks for PSC and $\mathrm{PTSO}_{\mathrm{syn}}$ are superceded by the corresponding checks over NVM, since the memory models PSC and $\mathrm{PTSO}_{\mathrm{syn}}$ are both supersets of NVM. That is, any trace over NVM must also be a trace PSC and $\mathrm{PTSO}_{\mathrm{syn}}$. For two transactions, two locations and two values, the checks for PMDK, PMDK-TML and PMDK-NOREC take 12.16, 17.36, and 56.02 seconds, respectively.

1.3 Summary of Results

Fig. 2 summarises our experiments on the upper bound checks, where the times shown combine the compilation and model exploration times. Each row represents an experiment that bounds the number of transactions (#txns), locations (#locs), values (#val) and the size of the persistency and store buffers (#buff). The times reported are for an Apple M1 device with 16GB of memory. The first row depicts a set of experiments where the implementations execute directly on NVM, without any buffers. As we discuss below, these tests are sufficient for checking lower bounds. The baseline for our checks sets the value of each parameter to two, and Fig. 2 allows us to see the cost of increasing each parameter. Note that all models time out when increasing the number of transactions to three, thus these times are not shown. Also note that for TXPMDK (which is single-threaded), the checks for PSC also cover $\mathrm{PTSO}_{\mathrm{syn}}$, since $\mathrm{PTSO}_{\mathrm{syn}}$ is equivalent to PSC in the absence of races [6]. Nevertheless, it is interesting to run the single-threaded experiments on the $\mathrm{PTSO}_{\mathrm{syn}}$ model to understand the impact of the memory model on the checks.

In our experiments we use FDR4's built-in *partial order reduction* features to make the upper bound checks feasible. This has a huge impact on the model checking speed; for instance, the check for PMDK-TML with two transactions, two locations, two values and buffer size of two reduces from over 6000 seconds

(1 hour and 40 minutes) to under 7 seconds, which is almost a 1000-fold improvement! This speed-up makes it feasible to use FDR4 for rapid prototyping when developing programs that use TXPMDK, even for the relatively complex PTSO_{syn} memory model.

References

1. Derrick, J., Doherty, S., Dongol, B., Schellhorn, G., Travkin, O., Wehrheim, H.: Mechanized proofs of opacity: a comparison of two techniques. Formal Aspects Comput. **30**(5), 597–625 (2018). https://doi.org/10.1007/s00165-017-0433-3
2. Dongol, B., Derrick, J.: Verifying linearisability: A comparative survey. ACM Comput. Surv. **48**(2), 19:1–19:43 (2015). https://doi.org/10.1145/2796550
3. Dongol, B., Le-Papin, J.: Checking opacity and durable opacity with FDR. In: Calinescu, R., Pasareanu, C.S. (eds.) SEFM. Lecture Notes in Computer Science, vol. 13085, pp. 222–242. Springer (2021). https://doi.org/10.1007/978-3-030-92124-8_13
4. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: FDR3 — A Modern Refinement Checker for CSP. In: Ábrahám, E., Havelund, K. (eds.) TACAS. Lecture Notes in Computer Science, vol. 8413, pp. 187–201 (2014)
5. Hoare, C.A.R.: Communicating sequential processes (reprint). Commun. ACM **26**(1), 100–106 (1983). https://doi.org/10.1145/357980.358021
6. Khyzha, A., Lahav, O.: Taming x86-tso persistency. Proc. ACM Program. Lang. **5**(POPL), 1–29 (2021). https://doi.org/10.1145/3434328
7. Lowe, G.: Analysing lock-free linearizable datatypes using CSP. In: Gibson-Robinson, T., Hopcroft, P.J., Lazic, R. (eds.) Concurrency, Security, and Puzzles - Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 10160, pp. 162–184. Springer (2017). https://doi.org/10.1007/978-3-319-51046-0_9
8. Raad, A., Lahav, O., Wickerson, J., Balcer, P., Dongol, B.: Intel PMDK transactions: Specification, validation and concurrency (Artifact) (2024). https://doi.org/10.6084/m9.figshare.24988173.v1