DTHREADS: Efficient Deterministic Multithreading

Robert Hoff

November 22, 2011

1 Summary

The paper [4] introduces `DTHREADS`, a deterministic version of the familiar pthreads (POSIX threads). Determinism is important for producing reliable software, non-deterministic software may for instance produce rare bugs that are hard to find when testing. dthreads is a user-space implementation that preserves the exact semantics and API of phtreads, it is therefore easy to use and requires no special hardware. Because of the user-space approach however, software adhere to conventions to achieve predictable behaviour, and so is vulnerable to buggy code that doesn't comply. In tests dthreads runs faster than comparable deterministic systems, and usually it is almost as fast as pthreads.

2 The Problem

One use of threads is for utilising benefits of parallel computing, but implementations such as pthreads produce non-deterministic results because of timing dependencies in the compiled code. This is a difficult software production and deployment problem, particularly for debugging and testing. In many systems today a difficult problem of replay debugging is used that requires logging every memory access.

One line of investigation is to eliminate non-determinism all-together, thus enforcing multi-threaded code to always have the same effect (i.e. observable output). This introduces shared-memory determinism, a currently active research area which is tackled by various approaches such as CoreDet [3]. This system can be improved because it doesn't distinguish between thread-local and global reads and writes to memory, and it serialises external library calls (it is more efficient to trust the library provider).

3 Solution

Non-deterministic results may occur by race conditions, deadlocks, atomicity or order violations. It was shown by Devietti *et al.*[2] that the problem can be reduced to ensure that the order of shared-memory accesses are the same for each execution. In the dthreads implementation the 'threads' are replaced by 'processes' with their own memory space (and resources). Determinism is achieved by monitoring shared memory accesses, called synchronisation points, and by switching the execution between two separate phases. In the parallel phase all the threads initially run, but each thread halts at different times when a synchronisation point is reached. At this point all the thread's separate memory models are written to the global state, and a serial phase begins where each threads runs in turn by a token-passing protocol. After all the threads have been resolved the parallel phase restarts.

4 Evaluation

In tests dthreads is usually twice as fast as CoreDet and usually almost as fast as phtreads. In the instances were dthreads is slower it may be attributed to many synchronisation operations cause frequent phase changes. There is a load imbalance problem when the system is waiting for straggling threads. Unfortunately an optimisation for this doesn't exist because reverting to the serial phase before all threads reach a synchronisation point would cause non-deterministic behaviour.

In some rare cases dthreads performs better than pthreads. the reason for this is that it avoids a conventional culprit, called false sharing. In phthreads slow memory synchonisations occur when two cores write within the same word boundary, but this is avoided in the dhtreads model.

5 My Opinion

dthreads is a software-only implementation that runs in user space, so is vulnerable to some types of code or ad-hoc compilation. For example if a thread is written so that it never reaches a synchronisation point, it would stall the entire process. And, any updates by a dthread to a stack variable would cause the program to fail, since this is regarded as a deprecated practise. This is unlike the Determinator system [1], where the support is provided as a kernel API. This approach has higher barriers though because it requires changes in the OS kernel.

It is a benefit that it is easy to use dthreads because it preserves exactly the same API as the existing pthreads, so one just needs to change the thread library. I'm wondering in order to use dthreads effectively, it would probably need to have a guarantee that future version produces the same

determinism, otherwise one may have to rewrite or test one's code again (or ensure that one continues to compile against consistent dthread version).

6 Questions

1. Considering the phread example the output of the code can produce (0,1), (1,0) and (1,1). In contrast dthreads always produce (1,1), but can we be sure the next version of dthreads will produce (1,1)? Also, is (1,1) better than (1,0), does it make any difference?

2. Can we be sure that non-determinism is not better solved higher up in the application chain, by writing code with pthreads that always produce the same result?

3. Are there other advantages of non-determinism, apart from exectution time, that may be lost by introducing determinism?

References

[1] Aviram A, Weng S C, Hu S, and Ford B. Efficient system-enforced deterministic parallelism. *USENIX*, 2010.

[2] Devieetti J, Lucia B, Ceze L, and Oskin M. Dmp: deterministic shared memory multiprocessing. *ACM ASPLOS*, 2009.

[3] Bergan T and et al. Coredet: A compiler and runtime system for deterministic multithreaded execution. *ACM ASPLOS*, 2010.

[4] Liu T, Curtsinger C, and Berger E D. Dthreads: Efficient deterministic multithreading. *ACM SOSP*, 2011.