Transactification of Real World System Libraries

Authors: Nebojša Miletić, Vesna Smiljković

Introduction

Transactional memory is a concurrency control mechanism analogous to database transactions for controlling access to shared memory resources in concurrent computing.
It is meant to simplify concurrent programming by allowing group of load and store operations to execute in an atomic way.
Its main intention is to replace the lock mechanism that is widely used and has showed as very hard to program with, but viable alternative has never appeared.

BSCMSRC conducts fundamental and applied research in TM by developing TM applications, benchmark suites, compiler and runtime tools and architectural simulators for TM.

What the research is?

Transictification is a process of converting lock-based software to one that uses transactional memories. In that way the converted application fully facilitates the semantics of TM. Sometimes it means modification of the code, and sometimes it is necessary to write certain functions from the scratch, but it is important to keep the same interface and funcionality.

Why a real world system library?

System library is a necessary tool to write other applications and only if it uses TM, it allows to compose applications which use TM, as well. Otherwise, lack of composability is a problem because it can lead to undefined behavior – what would happen if a transaction aborts after the lock and before the unlock command in the following code:

```
atomic {
    lock(resource);
    ...
    unlock(resource);
    ...
 }
```

Working with real library makes real applications possible and it gives homogeneous approach to the developers.

What the intended outcome is?

Ultimate goal is to have existing and community adopted **system library converted to equivalent transactional version**. In that way, other real applications can be used on top of it, without fear of undefined behaviours and with fully facilitated TM semantics.

In certain cases, some of the **locks are inherently necessary** (e.g. keyboard input, printer output etc). Such cases will have to be reasonably explained. In worst case, we would show that transactification of existing system library is possible at very high cost and try to recommend alternative solutions.

What are the challenges?

Semantic analysis vs. find/replace

It is not possible just to simply replace lock-unlock pair with atomic section.
- Sometimes they are not paired because they exist in different functions/files.
- Some locks are unlocked just by setting certain properties to certain values and vice-versa.
- Not every lock has corresponding unlock and vice-versa.

System calls

Syscalls are capable of modifying the state of the memory or the system in a way that their side effects cannot be reverted. So, when a malloc, write, thread_kill are called inside a transaction, there must be a solution for the case of abort. Strategies that can be used are:
- Undo functions (fig. 1)
- Defer syscall's action until the commit time (fig. 2)
- Irrevocable transactions
- Unrestricted transactions

lock-based

```
lock (thread)
threadsafe_malloc(...)
unlock (condition)
```

transactified

```
atomic {
  register_undo(f);
  threadsafe_malloc(...)
}
...
void f (...) {
  free(...)
}
```

Fig. 1: UNDO MECHANISM PSEUDOCODE

lock-based

```
lock (condition)
for (td in allThreads)
  td.restart()
unlock (condition)
```

transactified

```
thread[] toRestart;
atomic {
  for (td in allThreads)
    toRestart.push(td)
}
for (td in toRestart)
  td.restart()
```

Fig. 2: DEFERRAL MECHANISM PSEUDOCODE

Current work, next steps

At this moment we are **investigating** what are all possible problems that we might face, locating the most viable TM to use (together with other tools, e.g. for compiling). Very important decision will be the system library that we will work with, but it has to be wide adopted and familiar to the community.

After the transactification is finished, in order to prove the concept, the library will have to be **tested** on relevant benchmarks, but also with some TM and non-TM real applications. Hopefully, a publication will follow to make the TM step closer to the general programmers community.

BSC-Microsoft Research Centre · Barcelona Supercomputing Center, Centro Nacional de Supercomputación · UNIVERSITAT POLITÈCNICA DE CATALUNYA · Microsoft Research