

C-Sim version 5.0

Roman Jokl, Stanislav Racek
University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 22, Plzen
Czech Republic
rjokl@kiv.zcu.cz, stracek@kiv.zcu.cz

Abstract

The paper presents the C-Sim simulation tool that enables an experimental evaluation (i.e. testing) of parallel and distributed programs behavior using their close-to-reality C-language source code and a simulated operational environment.

1. Simulation based method of parallel programs testing

The method uses discrete-time process-oriented simulation model of a parallel or distributed computer system that executes a program $\Pi = \{P_i\}$. Source code of the program Π (that is composed from threads' or process' programs P_i) should be a part of the simulation program S source code. Due to the fact that a substantial set of parallel programs is C language written (using e.g. POSIX, PVM or MPI interface), this language seems to be a good implementation basis for the given purpose.

Every program P_i can be statically mapped to a Simula-like coroutine and every thread or process of the tested program can be dynamically mapped to a simulation process that is executed using discrete model-time basis. It means that real-time concurrent program execution is transformed into "interleaved" (i.e. pseudo-parallel) model-time execution. Points of a process execution switching can be recognized as an interaction procedure calls (e.g. I/O call, `lock()` or `unlock()` semaphore function calls, etc.). Every locally executed part of a process code (e.g. between `lock()` and `unlock()` calls) can be equipped with a pseudo-statement `hold(d)` where d is a model-time duration of the execution (either estimated to have a corresponding real-time duration value or generated randomly). The method was described in [5] and its substantial properties are as follows:

- Simulation based (serialized) execution can be made quite deterministic, even when (pseudo)random numbers generator is used to obtain duration d values. It offers a relatively easy way to remove program bugs.
- A submodel Q of a computation environment can be used as a part of simulation program (i.e. $S = \{\Pi, Q\}$), so a close-to-reality data sets or timed sequences can be used as the computation input.
- Model based repeated execution of a parallel program can evaluate both the control-flow correctness and logical correctness. But it *cannot prove* the correctness, compared with model-checking method using e.g. the SPIN model checker.
- Time responses of the computation can be evaluated. This is important e.g. for hard real-time embedded computer control systems and it is not possible to do that using a conventional model-checking method.
- There is only a minimal semantic gap between the parallel program and its evaluated model because C-language written source code serves both for the used algorithm specification (here a kind of executable specification) and implementation purposes.

2. C-Sim version 5.0 implementation principles

There are two implementations of pseudo-parallel computation in the C-Sim v. 5.0 library (what is the main improvement when compared with the previous version 4.1 - see [6]). The first one is based on the functions `setjmp()` and `longjmp()`, which are a part of the standard C-language libraries. They provide means for switching the process context without using any nonstandard operations, but their use causes invalidation of stack. This implies some restricting rules for a user of the library, particularly use of local (automatic) variables needs to be explicitly denoted

and switching of process context within a nested function is impossible. On the other side, thousands of processes can be effectively created within a simulation model.

The second method of implementation of the pseudo-parallel processes is based on the use of POSIX threads. A thread is created for each simulation process. C-Sim run-time control assures that only one thread is being executed at a time. The POSIX implementation is slower than long-jump implementation because the process switching in threads is more complex than the long-jump operation. On the other side this implementation removes the above stated limitations of its counterpart. Library user may choose desired implementation at compile time by defining a preprocessor macro on the command line.

The C-Sim v. 5.0 library provides additional modules usable for parallel algorithms testing. There is a Semaphore module that implements an integer semaphore and a Message Passing module that implements asynchronous message passing primitives. A qualified user can construct (as an additional program module) a set of proprietary operations that are close to a real communication interface (e.g. PVM or MPI). The model-time delay can be included inside the communication operations as well.

3. C-Sim utilization spectrum

Various characteristics of a distributed and parallel program can be validated using the C-Sim library. A deadlock-free control flow and time independent result of computation are often tested. It is also possible to evaluate performance parameters. In this case, it is necessary to estimate a duration of computing operations and communications.

There are examples at the website [1] that show possibilities of C-Sim utilization. Some of them are conventional models of queuing networks. The objective of these models is computation of a given queuing network performance parameters. The computation can be easily parallelized using e.g. PVM tool that has C language interface.

Another group of examples are demonstration models of simple parallel algorithms. The most simple of them models several processes that utilize a shared resource using a binary semaphore (i.e. a model of multithreading). Locking and unlocking operations are replaced by operations provided by C-Sim's Semaphore module. Time spent outside and inside the critical section is modeled by the `csim_hold()` pseudo-statement. To demonstrate a flavor of C-Sim based programming, we can show a sketch of program code of a simple cyclic process that alters local computation with a shared resource access:

```
for (;;) {  
    my.result = f (my.x, ...);  
    /* local computation */
```

```
    csim_hold (t1);  
    /* added duration of computation */  
    csim_lock_sem (p_sem);  
    /* locking global semaphore */  
    glob_result = g (my.result, ...);  
    csim_hold (t2); /* duration */  
    csim_unlock_sem (p_sem);  
}
```

As for `t1` and `t2` parameters of `csim_hold()` call, their construction depends on the purpose of computation. When we are interested in verifying flow correctness, it is convenient to construct parameter value as a pseudorandom number, say exponentially distributed. When we are trying to estimate a performance parameter, say the mean frequency of threads iterations, it is better to use a random number with Gaussian distribution.

C-Sim library can be used to test properties of distributed asynchronous algorithms as well. C-Sim library supports it by the Message Passing module that exports a kind of conventional `send()` and `receive()` operations for asynchronous communication. An example of distributed leader election algorithm serves as demonstration of this part of application spectrum.

The C-Sim library (version 4.1, long-jump implementation) has been tested thoroughly within the EU 5th framework program project FIT (Fault Injection for Time Triggered Architecture) [2], [4]. Large simulation model written in C-Sim was used (as one from several tools) to verify fault-tolerant properties of TTP/C protocol [3].

Every node of the modeled distributed system was described by means of a set of processes. The set includes several threads of the communication protocol itself and several threads of a part of application program that resides at the node. A model of the (doubled) communication bus involves two more processes. Simulated memory-disturbing transient faults were used in order to test specified system's properties (it needs additional processes that inject faults and observe their influence).

References

- [1] C-Sim homepage, <http://www.c-sim.zcu.cz>.
- [2] FIT homepage, <http://www.fit.zcu.cz>.
- [3] TTTech company homepage, <http://www.fit.zcu.cz>.
- [4] P. Herout, S. Racek, and J. Hlavicka. Model-based dependability evaluation method for TTP/C based systems. *Proc. European Dependable Computing Conference (EDCC-4, Toulouse, France)*, October 2002.
- [5] J. Hlavicka, S. Racek, and P. Herout. Evaluation of process controller fault tolerance using simulation. *Simulation Practice and Theory*, 7:769–790, March 2000.
- [6] R. Jokl. C-Sim version 5.0. *MSc. thesis, DCS/UWB Pilsen*, <http://www.c-sim.zcu.cz>, June 2001.