Useful Tips and FAQs for Using CKTSO in Circuit Simulators

1. Selection of `CKTSO_Factorize` and `CKTSO_Refactorize`

In DC simulation, `CKTSO_Factorize` is always recommended.

In transient simulation, the following method is recommended. Conventional SPICE-style simulators use the following method to judge whether Newton-Raphson iterations are *converged*

$$\left|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right| < \varepsilon_a + \varepsilon_r \times \min\{||\boldsymbol{x}^{(k)}||, ||\boldsymbol{x}^{(k-1)}||\}$$

where ε_a and ε_r are given absolute and relative thresholds, respectively. One can simply use the same method **with larger thresholds** to judge whether Newton-Raphson iterations are *converging*. If so, it means that the matrix values are changing slowly and `CKTSO_Refactorize` can be used; otherwise `CKTSO_Factorize` should be used. The thresholds for judging whether Newton-Raphson iterations are converging may be empirically determined. Note that the first factorization must call `CKTSO_Factorize`.

2. Fast factorization of `CKTSO_Factorize`

The last argument of `CKTSO_Factorize` specifies whether fast factorization is enabled. Please note that even if fast factorization is enabled, the first factorization cannot use fast factorization and is much slower than subsequent factorizations.

In DC simulation, there are some independent simulation "*methods*", like direct Newton-Raphson iterations, GMIN stepping, source stepping, and pseudo-transient. The first factorization of each "*method*" should disable fast factorization. To be more conservative, for each different stepping value in stepping-based methods, the first factorization may also disable fast factorization.

In transient simulation, generally fast factorization can always be used, with the exception that each time after the time node is rolled back due to divergence, the next factorization should also disable fast factorization.

3. Forced sequential solving of `CKTSO_Solve`

The second last argument of `CKTSO_Solve` specifies whether to force CKTSO to use sequential solving. Please note that parallel solving requires an initialization step that spends about 2-5X of sequential solving time. The initialization is needed once if the symbolic structure of the LU factors has been changed (re-pivoting has happened).

First of all, parallel solving have different performance for different matrix storage orders and on different systems. For column-mode matrices on Windows, parallel solving does not tend to get better performance so it is not recommended. For other situations parallel solving may get performance benefits, and whether to use forced sequential solving can be determined as follows.

In DC simulation, forced sequential solving is recommended.

In transient simulation, forced sequential solving should be set for

CKTSO_Factorize. For CKTSO_Refactorize, solving can be parallel. Please note that even if forced sequential solving is disabled, CKTSO may still use sequential solving, according to some internal strategies.

4. Ax values of CKTSO_Analyze

CKTSO_Analyze is usually a one-time work. If ax is provided when calling CKTSO_Analyze, the matrix ordering and static pivoting are both based on both the symbolic pattern and values of the matrix which are provided to CKTSO_Analyze. In circuit simulation iterations, the matrix values are changing. Hence, the values provided to CKTSO_Analyze should be representative and have similar features of the matrix values during the iterations of an entire simulation process. This can be done by adding a specialized value filling function which fills typical matrix values (for example, if a value is expected to frequently become 0 in circuit simulation iterations, it should be filled with 0). ax can be NULL when calling CKTSO_Analyze. In this case, the matrix ordering is completely determined based on the matrix structure, and static pivoting is invalid. However, **providing ax values is strongly recommended**, because a pure structure-based ordering may lead to a dramatic increase in the fill-ins, especially when the input matrix is stored in a quasi-random order.

5. How to select ordering method?

CKTSO has 10 ordering methods where methods 1-8 are minimum degree variants and methods 9-10 are nested dissection variants. Nested dissection ordering is good for large post-layout/mesh-like circuits, while minimum degree ordering is good for small pre-layout circuits.

By default, CKTSO selects the best ordering method from the 10 methods. However, the selection process is slow. If the input matrix is from post-layout simulations and is very large, then simply setting iparm[2]=11 will only use nested dissection ordering. If the input matrix is from pre-layout simulations and is very sparse, then setting iparm[2]=18 will select the best ordering method from 8 minimum degree variants.

6. Why does CKTSO_CreateSolver return -10 (not supported)?

CKTSO uses AVX2 and FMA instructions. CKTSO_CreateSolver checks whether the CPU supports such instructions. If not, a -10 code is returned. Please note that the check may fail on a *virtual machine* even if the CPU actually supports such instructions. In this case, CKTSO_CreateSolverNoCheck which bypasses the instruction check can be used.

7. Why is the first call of CKTSO_Factorize much slower than subsequent calls?

CKTSO integrates a pivoting-reduction technique which can significantly improve the performance and parallel scalability from the second factorization. In circuit simulation, since matrix values usually change smoothly during the Newton-Raphson iterations, in most cases the pivoting order is not changed so symbolic-related operations may be skipped from the second factorization. However, the first factorization needs to perform all necessary operations.

8. For parallel solving, why is the first call of `CKTSO_Solve` much slower than subsequent calls?

Before the first parallel solving process, an initialization step for the scheduler is needed. The scheduler depends on the symbolic structure of the LU factors. The initialization step is needed every time when the symbolic structure of the LU factors is changed (factorization with pivoting is called and the structure of the LU factors really changes). If the symbolic structure of the LU factors keep unchanged, the initialization step is skipped and the solving process may get its benefit from parallelism.

9. Does each linear system need to be processed by an individual solver instance?

No. A solver instance can handle multiple linear systems in turn. `CKTSO_Analyze` first frees data of the previous matrix, and then allocates memory for the new matrix.

However, if multiple linear systems need to be solved in parallel, multiple solver instances are needed. In this case, one point must be noted is that **each solver instance should run in an individual thread**. Multiple solver instances cannot run in parallel in a thread.