User Guide of CKTSO-GPU

(version 20221214)

Xiaoming Chen (chenxiaoming@ict.ac.cn)

CKTSO-GPU is a GPU acceleration module of CKTSO. The package can be seamlessly integrated with CKTSO. It provides GPU acceleration for re-factorization and solving using CUDA. Both real and complex matrices are supported. GPU-based re-factorization and solving can be invoked after factorization with pivoting has been performed on CPU.

CKTSO-GPU can accelerate slightly-dense matrices but cannot efficiently accelerate highly-sparse circuit matrices. For extremely-sparse matrices, just using CKTSO running on multicore CPUs can obtain the best performance. **CKTSO-GPU is recommended when flops/NNZ(L+U) is larger than 50.** In this case, CKTSO-GPU tends to perform faster than CKTSO (1 thread).

1. Functions

CKTSO-GPU provides 5 functions which are very easy to use. Both 32-bit and 64-bit (with `_L` in the function name) integer versions are provided. For the 32-bit integer version, only the indexes of matrix **A** use 32-bit integers, but the internal data structures for LU factors still use 64-bit integers.

(1) `CKTSO(_L)_CreateGpuAccelerator`

This function creates the CKTSO-GPU handle and also retrieves pointers to the input and output arrays. The target GPU is also set. Subsequent re-factorization and solving will be performed on the selected GPU.

(2) `CKTSO(_L)_InitializeGpuAccelerator`

This function initializes the internal data of CKTSO-GPU, based on the internal data of CKTSO. It can be called after factorization with pivoting is performed on CPU by CKTSO (i.e., `CKTSO(_L)_Factorize` has been called). It is recommended to sort the factors (i.e., calling `CKTSO(_L)_SortFactors`) before calling this function. Without sorting the factors by CKTSO, this function will sort the factors, but the performance is lower. Each time when the LU factors structure has been changed (i.e., `CKTSO(_L)_Factorize` has been called), this function should be recalled before subsequent re-factorization and solving on GPU.

(3) `CKTSO(_L)_DestroyGpuAccelerator`

This function frees any data associated with CKTSO-GPU and also destroys the handle.

(4) `CKTSO(_L)_GpuRefactorize`

This function performs re-factorization on GPU. The input matrix data `ax[]` is in host memory.

(5) `CKTSO(_L)_GpuSolve`

This function performs forward and backward substitutions on GPU. The input and output vectors `b[]` and `x[]` are both in host memory. Both row and column modes are supported. Only one right-side vector is supported. For multiple right-side vectors, please call this function multiple times.

2. Return Values

CKTSO-GPU uses the error codes of CKTSO. In addition, CKTSO-GPU also uses the following additional error codes.
* -51: insufficient GPU resources.
* -52: CKTSO-GPU has not been initialized.
* -53: matrix has not been re-factorized by GPU.
* If the return value is a positive integer, it means a CUDA runtime function error. For example, 2 indicates `cudaErrorMemoryAllocation`, and 35 indicates `cudaErrorInsufficientDriver`. Please refer to NVIDIA's CUDA documents to check the detailed error code.

3. Input Parameters

* `iparm[0]`: timer control. Zero disables the internal timer. A positive number enables the high-precision timer. A negative number enables the low-precision timer. Default is 0 (no timer).
* `iparm[1]`: threshold for distinguishing bulk and pipeline modes in re-factorization. Default is 128.
* `iparm[2]`: factors array allocation ratio (percentage). CKTSO-GPU allocates a bit more memory for the LU factors. If the LU factors have been changed and the size is still within the allocated length, recalling `CKTSO(_L)_InitializeGpuAccelerator` will not reallocate the memory. Default is 110, which means 1.1.
* `iparm[3]`: whether to reallocate memories when size reduces when recalling `CKTSO(_L)_InitializeGpuAccelerator`. Default is 0 (reusing previous memories).
* `iparm[4]`: number of blocks per multiprocessor, for re-factorization. 0 means automatic decision by CKTSO-GPU. Default is 0 (automatic decision).
* `iparm[5]`: number of threads per block, for re-factorization. 0 means automatic decision by CKTSO-GPU. Default is 0 (automatic decision).
* `iparm[6]`: number of blocks per multiprocessor, for solving. 0 means automatic decision by CKTSO-GPU. Default is 0 (automatic decision).
* `iparm[7]`: number of threads per block, for solving. 0 means automatic decision by

CKTSO-GPU. Default is 0 (automatic decision).

4. Output Parameters

* `oparm[0]`: time of `CKTSO(_L)_InitializeGpuAccelerator`, in microsecond.
* `oparm[1]`: time of `CKTSO(_L)_GpuRefactorize`, in microsecond.
* `oparm[2]`: time of `CKTSO(_L)_GpuSolve`, in microsecond.
* `oparm[3]`: host memory usage, in bytes. It only reports the host memory usage of the CKTSO-GPU instance, excluding the CKTSO instance.
* `oparm[4]`: GPU global memory usage, in bytes.
* `oparm[5]`: host memory requirement, in bytes, when -4 is returned (for the last malloc/realloc failure). It is only related to the CKTSO-GPU instance.
* `oparm[6]`: GPU memory requirement, in bytes when `cudaErrorMemoryAllocation` (i.e., 2) is returned (for the last `cudaMalloc` failure).

5. System Requirements

The provided libraries are compiled on CentOS 7.9 with gcc 4.8.5 and CUDA 11.0. They require **NVIDIA driver 450.51.05 or higher**.

Supported computing capabilities of CUDA: 3.5, 3.7, 5.0, 5.2, 6.0, 6.1, 7.0, 7.5, and 8.0.

Current CKTSO-GPU is compatible with **CKTSO version 20221207 or higher**.

6. Notes

* CKTSO-GPU uses ***the row-major order by default***. If the matrix is stored in the column-major order, please set the last argument of `CKTSO(_L)_GpuSolve` to `true`. Column-wise solving tends to perform slower than row-wise solving.
* On some GPUs, it is found that L1 cache must be disabled to get correct results. Thus, libraries with L1 cache disabled are also provided. The overhead is some performance loss (about ~20%).
* Scaling (`iparm[7]` of CKTSO) is not supported on GPU.
* The matrix type (real or complex) of CKTSO-GPU follows CKTSO. Specifically, it depends on the argument `is_complex` of `CKTSO(_L)_Analyze`.