User Guide of CKTSO-GPU

(version 20221123)

Xiaoming Chen (chenxiaoming@ict.ac.cn)

CKTSO-GPU is a GPU acceleration module of CKTSO. The package provides GPU acceleration for re-factorization and solving using CUDA. It can be called after factorization with pivoting has been performed on CPU.

CKTSO-GPU can only accelerate slightly-dense matrices but a majority of circuit matrices cannot be accelerated. For those highly-sparse matrices, just using multicore CPUs can obtain the best performance. **CKTSO-GPU is recommended when flops/NNZ(L+U) is larger than 200-300.**

1. Functions

CKTSO-GPU provides 5 functions which are 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_CreateGpuAccelerator/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 are performed on the selected GPU.

(2) `CKTSO_InitializeGpuAccelerator/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_Factorize/CKTSO_L_Factorize` has been called). Each time when LU factors structure has been changed (i.e., `CKTSO_Factorize/CKTSO_L_Factorize` has been called), this function must be recalled before subsequent re-factorization and solving on GPU.

(3) `CKTSO_DestroyGpuAccelerator/CKTSO_L_DestroyGpuAccelerator`

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

(4) `CKTSO_GpuRefactorize/CKTSO_L_GpuRefactorize`

This function performs re-factorization on GPU.

(5) CKTSO_GpuSolve/CKTSO_L_GpuSolve

This function performs forward and backward substitutions on GPU. Only the row mode is supported. **For a matrix stored in the column order, please use the transposed mode of CKTSO (i.e., setting `row0_colposi_trannega` to a negative value when calling `CKTSO_Analyze/CKTSO_L_Analyze`)**, and in this case, CKTSO-GPU will treat the matrix as row-order stored and perform transposition in every re-factorization. Currently only 1 right-side vector is supported. For multiple right-side vectors, please call this functions 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 means cudaErrorMemoryAllocation. Please refer to NVIDIA's CUDA documents to check the error code.

3. Input Parameters

* iparm[0]: timer control. 0 disables the internal timer. A positive number enables the high-precision timer. A negative number enables the low-precision timer. Default is 0.
* 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_InitializeGpuAccelerator/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_InitializeGpuAccelerator/CKTSO_L_InitializeGpuAccelerator. Default is 0.
* iparm[4]: number of blocks per multiprocessor, for re-factorization. 0 means automatic decision by CKTSO-GPU. Default is 0.
* iparm[5]: number of threads per block, for re-factorization. 0 means automatic decision by CKTSO-GPU. Default is 0.
* iparm[6]: number of blocks per multiprocessor, for solving. 0 means automatic decision by CKTSO-GPU. Default is 0.
* iparm[7]: number of threads per block, for solving. 0 means automatic decision by CKTSO-GPU. Default is 0.

4. Output Parameters

* `oparm[0]`: time of `CKTSO_InitializeGpuAccelerator/CKTSO_L_InitializeGpuAccelerator`, in microsecond.
* `oparm[1]`: time of `CKTSO_GpuRefactorize/CKTSO_L_GpuRefactorize`, in microsecond.
* `oparm[2]`: time of `CKTSO_GpuSolve/CKTSO_L_GpuSolve`, in microsecond.
* `oparm[3]`: host memory usage, in bytes. It only reports the memory usage of the CKTSO-GPU handle.
* `oparm[4]`: GPU memory usage, in bytes.
* `oparm[5]`: host memory requirement, in bytes, when -4 is returned (for the last malloc/realloc failure).
* `oparm[6]`: GPU memory requirement, in bytes when `cudaErrorMemoryAllocation` (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.

CKTSO-GPU is compatible with **CKTSO version 20221123 or higher**.

On some GPUs, it is found that using L1 caches can lead to incorrect results. Thus, libraries with L1 cache disabled are also provided. The overhead is some performance loss.