2.[st] Task in *HPC-I*

Deadline: Jan 5, 2024, 11:59pm

PDE solvers using CUDA

We are going to investigate and implement various solvers for linear systems of equations on GPU.
One long-distance goal consists in a potential incorporation as GPU-solver for (non-linear) systems of equations in the CARP context

Download first:

- the simple conjugate gradients solver[1] together with input data[2] (2.9 GB),
- the NVIDIA example[3] for direct solvers,
- my book[4] as with some algorithms in §6.

---

0. Given Code The directory of the cg solver contains two implementations:

(A) *cg_example.cpp*: The cg solver (git[5]) with ICC-preconditioning by NVIDIA from its library samples[6].

- See lines 2,3 how to compile and start the code, or run `make special`
- Check the cg implementation by NVIDIA, I suspect an algorithmic error in lines 367-373, see also Alg. 6.19 in the book.

(B) *main.cpp*: My cg for further work.

- `make run`
- `./main.NVCC_ data/square_100_n` uses other input files with $n \in [0, 7]$.
- cg with diagonal preconditioning is implemented in the CPU as well as in the GPU part.

We should have 3 variants for preconditioning in (B) at the end of 2.:

- identity (no preconditioning) $\underline{w} = \underline{r}$,
- diagonal pc $\underline{w} = D^{-1}\underline{r}$,
- IC (incomplete[7] Cholesky decomposition[8]) $LDL^T\underline{w} = \underline{r}$

[1] `http://imsc.uni-graz.at/haasegu/Download/cg.zip`
[2] `http://imsc.uni-graz.at/haasegu/Download/Math2CPP_data.zip`
[3] `http://imsc.uni-graz.at/haasegu/Download/cusolver_examples-main.zip`
[4] `https://imsc.uni-graz.at/haasegu/Lectures/Master_HPC/textbook.pdf`
[5] `https://github.com/NVIDIA/CUDALibrarySamples/tree/master/cuSPARSE/cg`
[6] `https://github.com/NVIDIA/CUDALibrarySamples`
[7] `https://en.wikipedia.org/wiki/Incomplete_Cholesky_factorization`
[8] `https://en.wikipedia.org/wiki/Cholesky_decomposition`

1. Check/compare the cg solvers, add IC to (B) [$\longrightarrow$ BV+FM]

(i) Write an additional constructor for `CRS_Matrix_GPU` that reads the date via the *mtx* files from (A).

(ii) Compare iteration history and run time of the cg algorithms in (A) and (B) on GPU without preconditioning.

(iii) Check the cg implementation by NVIDIA (A), I suspect an algorithmic error in lines 367-373, see also Alg. 6.19 in the book.

(iv) Transfer the setup and application of the IC-preconditioning from (A) to (B).

(v) Compare iteration numbers as well as run time with data from (A) and from (B).

2. Improve/compare the cg solver (B) [$\longrightarrow$ ZR+MK]
Tasks for implementational variants of diagonal preconditioned cg in (B).
Use also `nsys-ui ./main.NVCC` for profiling.

(vi) Compare run time for unified memory allocation (`cudaMallocManaged`) versus exclusive device memory allocation (`cudaMalloc`).

(vii) Substitute the cuBLAS routines in cg by your own kernel calls and compare the run time. This will require to use file suffix *.cu* instead of *.cpp*.

(viii) Combine computation of $\underline{r}$, $\underline{w}$ (, $\underline{u}$) and σ in one kernel call. What about the resulting run time? Take care for register spilling.
Is some overlap via different streams possible $\underline{u}$ vs. the other vectors?

(ix) Use your own kernel call for the sparse matrix product calculating $\underline{v}$ (timing regarding the cuSPARSE call) and combine that kernel with the inner product calculation $(\underline{s}, \underline{v})$.

3. Other iterative solvers and improvements. On basis of your version of (B):

(x) Can the IC preconditioner be separated into a setup step (memory allocation, pattern of L) and an update step (recalculation of entries in L).
This would be interesting for non-linear solvers.

(xi) Implement a gmres solver, Alg. 6.21 in the book.

- Notice that dense matrix H is dynamically growing with each iteration.
- Can the reallocation of H as well as the Givens rotation be overlapped with some other computations GPU?

To be continued (next page)

The html-online material by NVIDIA is more a brief introduction into the user interface than a full description of all function call in a library, see the appropriate pdf-links for the latter one.

- cuSOLVER[9]:
  - Dense and sparse matrices (CSR).
  - QR/LU/Cholesky factorization on (multiple) GPUs.
  - Re-factorization with the same sparsity pattern.
  - Bottleneck: Fill in causes large memory requirements and increased run time.
- cuSPARSE[10]:
  - Supports several sparse formats (incl. CSR) and dense format, see §6.5[11].
  - BLAS 1-3 functionality for sparse matrices.
    Especially multiplication of two sparse matrices, see cusparseSpGEMM[12] and cusparseSpGEMMreuse[13].
  - ICC(0) and ILU(0) are implemented for CRS-matrices, see §11 in cuSPARSE-Docu[14] and a preconditioner primer by Nvidia[15].
  - Re-factorization with the same sparsity pattern might be possible, look for cusparse<t>csrilu02_analysis() and cusparse<t>csrilu02().
- AmgX[16]:
  - Algebraic multigrid solver and preconditioner.
  - Krylov solvers (cg, g,res etc.) available.
  - MPI and OpenMP support.
  - Example[17].
  - Has to be installed separately from the AMG repository[18].
- cuDSS[19]
  - Direct sparse solver library.
  - LU/Cholesky factorization with 3 stages of factorization process:
    symbolic factorization, numeric factorization, solving step which are controlled via cudssPhase_t[20] in the cudssExecute()[21] function.
  - API simular to cuSPARSE.
  - Still in development (preview).
  - Has to be installed[22] separately.

[9] https://docs.nvidia.com/cuda/cusolver/index.html
[10] https://docs.nvidia.com/cuda/cusparse/
[11] https://docs.nvidia.com/cuda/cusparse/#sparse-matrix-apis
[12] https://docs.nvidia.com/cuda/cusparse/#cusparsespgemm
[13] https://docs.nvidia.com/cuda/cusparse/#cusparsespgemmreuse
[14] https://docs.nvidia.com/cuda/pdf/CUSPARSE_Library.pdf
[15] https://docs.nvidia.com/cuda/incomplete-lu-cholesky/index.html
[16] https://github.com/NVIDIA/AMGX
[17] https://developer.nvidia.com/amgx
[18] https://github.com/NVIDIA/AMGX
[19] https://developer.nvidia.com/cudss
[20] https://docs.nvidia.com/cuda/cudss/functions.html#cudsscreate
[21] https://docs.nvidia.com/cuda/cudss/functions.html#cudssexecute
[22] https://developer.nvidia.com/cudss-downloads

References

[HaSh19] Jaegeun Han and Bharatkumar Sharma. *Learn CUDA Programming.* Packt> (2019); buy[23]; download code[24], download figures[25]

[NSLS14] Dan Negrut, Radu Serban, Ang Li and Andrew Seidl. *Unified Memory in CUDA 6: A Brief Overview and Related Data Access/Transfer Issues.* TR-2014-09[26] (2014)

[ChAn17] Andrzej Chrzeszczyk and Jacob Anders *Matrix computationson the GPUCUBLAS, CUSOLVER and MAGMA by examples.* NVIDIA (2017); download book[27].

[23] https://www.packtpub.com/eu/application-development/cuda-cookbook
[24] https://github.com/PacktPublishing/Learn-CUDA-Programming
[25] https://static.packt-cdn.com/downloads/9781788996242_ColorImages.pdf
[26] https://www.researchgate.net/publication/326920953_Unified_Memory_in_CUDA_6_A_Brief_Overview_and_Related_Data_AccessTransfer_Issues
[27] https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf