GPU-BA: Bundle Adjustment on the GPU

Ronny Hänsch, Igor Drude
Technische Universität Berlin

----1. Intro----

For large-sized optimization problems, bundle adjustment has become more relevant over the past years. Several research papers were published, discussing the task of working with big amounts of data, retrieved from various internet photo collections. However, only a few work has been done considering the acceleration properties of GPU systems. At the moment, it is a common approach to address aforementioned problems with the "Levenberg-Marquardt" - algorithm, combined with the conjugate-gradients method for solving the normal equations. This thesis also studies other approaches than that, such as the nonlinear conjugate-gradient or the alterning resection-intersection procedure. Whilst the latter one reveals partly competitive performance on a GPU system, this thesis also explores other ideas to improve the convergence behavior, e.g. a hybrid between the Resection-Itersection and the Levenber-Marquardt - algorithm or the recently suggested embedded point iterations. Furthermore, this elaboration displays the possibilities of time- and space savings, when fitting the implementation strategy to the terms and requirements given by the specific assignment of realizing a bundler on a heterogeneous CPU-GPU system.

----2. Parametrization----

MBA uses the same parametrization as BAL (http://grail.cs.washington.edu/projects/bal/). Points are expressed as a 3D-vector. For cameras the pinhole model is used. More precisely, the extrinsic parameters involve 3 values for the rotation und 3 values for the translation. The rotation is compressed as a rodrigues rotation vector. The intrinsic parameters are reduced to the focal length. Further, radial distortion with the two parameters k1 and k2 is supported. The projection into the image space is computed as following:

$$(1) \quad P = R \cdot X + t$$
$$(2) \quad p = -P / P_z$$
$$(3a) \quad r(p) = 1.0 + k1 \cdot \|p\|^2 + k2 \cdot \|p\|^4$$
$$(3b) \quad p' = f \cdot r(p) \cdot p$$

Where R is the rotation matrix, computed from the rodrigues rotation vector, X is the 3D-point, t is the translation vector, f is the focal length and P.z is the third (z-) coordinate of the Point P (Point x moved to camera coordinate system). Due to the limitations of generic coding on GPU systems, MBA does not support the use of custom parametrizations. That means, the problem should be (possibly) reformulated, if you want to use MBA.

----3. Interface & Input----

There are two possibilities to change the configuration parameters for the solver algorithms. Either the user changes the arguments in the driver.cpp source file (requires new compilations after every change) or in the configuration file. The path to the configuration file should be given to the application via the arguments list behind the launch command as the first value (argv[1]).

The path parameter declares the path to the input file defining the problem. The structure of the input file should follow the same patterns as demanded by the BAL datasets.

Specifically, the input file should look like that:

<num_cameras> <num_points> <num_observations>
<camera_idx_1> <point_idx_1> <x_1> <y_1>
...
<camera_idx_num_obs> <point_idx_num_obs> <x_n_obs> <y_num_obs>
<camera_1>
...
<camera_num_cameras>
<point_1>
...
<point_num_points>

The order of the measurements is arbitrary. The order of cameras and points should be ascending, accordingly as they are referenced by the measurements.

----4. Output----

No output data is written. The user is encouraged to implement an own writer function for the final parameter vector, since there are no standards for doing this. After the optimization has finished (or during) some informations are printed onto the standard output (usually the shell/console). These informations include i.a. the time (t=), number of iteration (#iter), amount of completed cg-iterations (cgi:), mean squared error (MSE=), relative error decrease (rd:) and the information about the iteration beeing successfull (d?:) error after each iteration.

----5. Requirements----

To be able to execute MBA it is mandatory to use a system running on a CUDA enabled Nvidia- GPU. This GPU should at least support compute capability 2.0 (see https://developer.nvidia.com/cuda-gpus to find a list of CUDA enabled GPUs). Moreover, this program was developed with the CUDA TOOLKIT 6.5, which means that a lower version should not be used when compiling MBA. Only the two cuda libraries "cuda.lib" and "cudart.lib" are used. Another important note is, that MBA was only tested on windows systems. No windows specific libraries were included and the timer function should work on win32 and linux systems. Nevertheless, since the programm was not tested on other systems an adjustment in the source code could be necessary.

----6. Configuration settings----

The configuration file helps the user to control the program more convenient. This section shows the all possible parameters and a short description to each one. An example file is featured in the project folder.

-BAmode
    Determines which input parameters should be optimized.
    "metric" = optimize points and extrinsic camera parameters only.
    "focal_radial" = optimize focal length and radial distortion as well.

-Solver
    Choose the solver algorithm here.
    "LMA" = use the Levenberg-Marquardt algorithm
    "NCG" = use the nonlinear Conjugate Gradient algorithm
    "RI" = use the Resection-Intersection algorithm

-path
    Specify the path to the input file here. Don't use quotation marks.

-degeneracy_fix
    "true" = Fix cameras or points, which are to close the cameras. (Recommended)
    "false" = Do not manipulate the input.

-del_proj
    "true" = Delete outlier measurements.
    "false" = Do not delete measurements, but move the corresponding cameras a little bit.

-reorder
    "true" = Change measurements order to optimize the memory accesses.
    "false" = Do not change the order.

-print_progress
    "true" = Print time and error during the optimization.
    "false" = Do not print time and error during the optimization.

-trace
    "true" = Trace the progress and show state of each iteration after the optimization finished.
    "false" = Do not trace anything.

-show_min_sse
    "true" = Displayed sse is the currently lowest reached error in the optimization.
    "false" = Displayed sse is the actually computed error in this iteration.

-gradM_stop
    (float value) Specifies a stopping criteria threshold for too small gradient magnitudes.
    Optimization is stopped when norm(g) < gradM_stop, where g is the gradient.

-mse_stop
(float value) Specifies a stopping criteria threshold for a sufficiently small mean squared error.
Optimization is stopped when mse < mse_stop.

-diff_stop
(float value) Specifies a stopping criteria.
Stopped when abs(newSSE/(float)sse - 1) << diff_stop (too small error decrease).
Has to be very small, or the algorithm might stop too early.

-maxI
(int value) Specifies the max. numbers of iterations.

-tau
(float value) Specifies the starting value for the damping factor lambda.

-maxL
(float value) Specifies the max. value allowed for lambda.

-minL
(float value) Specifies the min. value allowed for lambda.

-cg_maxI
(int value) Specifies the max. numbers of cg-iterations.

-cg_minI
(int value) Specifies the min. numbers of cg-iterations.

-n_fs
(float value) Specifies a stopping criteria for the (linear) CG-Solver.
Stopped after iteration i when || r0||/||ri|| < n_fs^2.

-diagdamping
"true" = Use multiplicative diagonal damping.
"false" = Use additive damping by using only the first diagonal element of the hessian-matrix.

-schur
"true" = Use the schur complement trick.
"false" = Don't use the schur complement trick.

-epi
(int value) Specifies the max. numbers of embedded point iterations.
Use 0, if you don't want to use EPI.

-backsub
"true" = Perform the backsubstitution after solving the reduced camera system.
"false" = Don't perform the backsubstitution after solving the reduced camera system.

-hyb_switch
(float value) Specifies a switching criteria.
Switched to LMA when abs(newSSE/(float)sse - 1) < hyb_switch) (too small error decrease).
Use 0 if you don't want to use the hybrid solver.

-b_reset
(int value) Resets the NCG optimizer (set b=0) every b_reset iterations.

-start_a
(float value) Specifies the starting value of alpha for the line search.

-t
(float value) Specifies the step length for the line search (a_new = a*t).

-c1
(float value) Specifies the first wolfe condition parameter for the armijo rule 0<c1<1.

-c2
(float value) Specifies the second wolfe condition parameter for the curvature rule, 0<c1<c2<1,
c=0 → don't use curvature rule.

-inner_loops
(int value) Specifies the max. amount of inner iterations for the Resection-Intersection method.

----7. Structure of the source code----

This section enumarates all source files of the project „GPU-BA“ and describes their functions.

driver.cpp
This file includes the main function and hence the entry point to the program. In this function the congifuration and the input file are read. Then, after short preprocessing step, the Optimizer (specified in the configuration file) is started.

Datahandler.cpp
This file implements the functions, which are necessary for setting up the optimization.
The core functions are

LoadBundlerModelBAL :	Loads the input file into RAM.
parseXXConfig :	Loads the configurations.
PreProcessing :	Degeneration fix and reordering.

MBATimer.h
Provides timer functions in milliseconds precision for Windows and Unix systems.

NCGalgorithm.cpp
The source file contains two functions: the constructor and the main loop of the nonlinear conjugate gradients algorithm.

RI.cpp
The source file contains two functions: the constructor and the main loop of the resection-intersection algorithm.

LMalgorithm.cpp
The source file contains two functions: the constructor and the main loop of the Levenberg-Marquardt algorithm.

CalcUtilsGPU.cu
This file contains the core implementations and cuda kernels for several functions, which are used by the three algorithms. There are three regions in this file:

init: Contains the functions needed to initiate the algorithms and the GPU.
cuda kernels: Contains all the private cuda kernels, performing all the core computations.
interface: Contains the function calls used multiple times by different extern objects and therefore acts as an interface for the GPU computation calls.
Since this file contais approx. 40 functions there will be no detailed explanation for each one in this document. See the source code comments for more details.

Inverse.cuh
This file provides batched inversions for a large amount of 3x3, 6x6 and 9x9 matrices implemented as cuda kernles.

Solve.cuh
This file provides batched solvers (via Cholesky-factorization followed by a forward- and backward substitution) for a large amount of 3-, 6- and 9-dimensional symmetric linear systems implemented as cuda kernles.