# Image Segmentation:
# the Blake-Zisserman functional approach

December 22, 2016

## 1 Introduction

Source package is composed of two folders:

- `CPP` contains C++ source code and Makefile;
- `matlab` contains Matlab® functions for data creation (see Section 4).

### 1.1 C++ code compilation

Provided code supports sparse structured matrix-vector routines through cusp[1] library, that in turns depends on thrust[2]. To leverage the compilation process, both libraries are provided as source code in `CPP/include` subfolder. Package `Makefile` is already configured to compile required (and provided) targets, so in order to compile the full package, enter `CPP` folder and issue the command:

```
make
```

Upon successful compilation, `CPP/bintest` folder is created and contains a number of executables. Default compiler is `g++`, but different compilers can be selected. For example, if Intel® C++ compiler is present, change line 10 of `Makefile` and substitute `g++` with `icpc`.

Complete source code documentation can be obtained through `doxygen`[3], `Makefile doc` target can help on documentation building process:

```
make doc
```

Upon successful documentation generation, `CPP/html` folder is created and contains a number of html files, start by opening `index.html` with a web browser. To remove all generated executables and documentation files, use:

```
make cleanall
```

## 2 BCDAfullimage_test

BCDAfullimage_test will perform segmentation using BCDA algorithm.

### 2.1 Command line parameters

A call to program `BCDAfullimage_test` should adhere to:

```
BCDAfullimage_test tag1=value tag2=value ... inputFile
```

where `inputFile` is a binary file obtained by calling to MATLAB function `writeData()` (see Section 4 for details). Parameters are read from left to right, as a consequence, the following command:

```
BCDAfullimage_test objbinfile=mypars e=1e-3 myimage
```

will force BCDAfullimage to read all objective function parameters from objbinfile, and then to forget $\epsilon$ value read from file and use instead $\epsilon = 10^{-3}$.

Objective function tags:

**e** : $\epsilon$, default value: $10^{-2}$;

**d** : $\delta$, default value: 1.0;

**a** : $\alpha$, default value: 2.0;

**b** : $\beta$, default value: 1.0;

**mu** : $\mu$, default value: 1.0;

**tx** : $t_x$, default value: 1.0;

**ty** : $t_y$, default value: 1.0;

**xe** : $\xi_e$, default value: $0.01^{1/4}$;

**oe** : $o_\epsilon$, default value: $1e-4$;

**objbinfile** : binary file contatining objective function parameters.

Solver tags:

**fval_tol** : tolerance for stopping criterion on relative decrease of objective function, default value: $10^{-3}$;
**pcg_tol_sz** : stop tolerance in inner loop (**s** and **z**): when 0, adaptive tolerance value is used, default value: 0;
**pcg_tol_u** : stop tolerance in inner loop (**u**): when 0, adaptive tolerance value is used, default value: 0;
**ext_maxit** : maximum number of iterations allowed in BCDA outer loop, default value: 1000;
**pcg_maxit_sz** : pcg maximum number of iterations allowed in inner loop (**s** and **z**), default value: 1000;
**pcg_maxit_u** : pcg maximum number of iterations allowed in inner loop (**u**), default value: 1000;
**prec** : preconditioner type for pcg when solving for **u** (D or BD), default value: D;
**verbosity** : if 1, prints external loop info, if 2 adds also **u**, **s**, **z** loops info, default value: 0.

Main tags:

**printpars** : If greater than 0, prints all the parameters, default value: 0;
**outputFileTemplate** : template for output file names, default value: empty;
**initial_u** : file containing initial value for **u**, default value: empty;
**initial_s** : file containing initial value for **s**, default value: empty;
**initial_z** : file containing initial value for **z**, default value: empty.

# 3 OPARBCDAManager_test and OPARBCDAThreadManager_test

OPARBCDAManager_test will perform tiled segmentation using OPARBCDA algorithm.

## 3.1 Command line parameters

A call to program `OPARBCDAManager_test` should adhere to:

```
OPARBCDAManager_test tag1=value tag2=value ... inputFile
```

where `inputFile` is a binary file obtained by calling to MATLAB function `writeData()` (see Section 4 for details).

Objective function tags:

**e** : $\epsilon$, default value: $10^{-2}$;
**d** : $\delta$, default value: 1.0;
**a** : $\alpha$, default value: 2.0;

**b** : $\beta$, default value: 1.0;
**mu** : $\mu$, default value: 1.0;
**tx** : $t_x$, default value: 1.0;
**ty** : $t_y$, default value: 1.0;
**xe** : $\xi_e$, default value: $0.01^{1/4}$;
**oe** : $o_\epsilon$, default value: $1e-4$;
**objbinfile** : binary file contatining objective function parameters.

Solver tags:

**over_maxit** : maximum number of iterations allowed in OPARBCDA outer loop, default value: 10;
**over_tol** : tolerance for stopping criterion on relative decrease of objective function, default value: $10^{-3}$;
**over_lb** : value for target objective function value stopping criterion, default value: minimum double precision value used in running architecture (around $-1.8 * 10^{+308}$).
**verbosity** : if 1, prints external loop info, if 2 adds also **u**, **s**, **z** loops info, default value: 0.

Internal Solver tags:

**pcg_tol_sz** : stop tolerance in inner loop (**s** and **z**): when 0, adaptive tolerance value is used, default value: 0;
**pcg_tol_u** : stop tolerance in inner loop (**u**): when 0, adaptive tolerance value is used, default value: 0;
**ext_maxit** : maximum number of iterations allowed in BCDA-like loop, default value: 1000;
**pcg_maxit_sz** : pcg maximum number of iterations allowed in inner loop (**s** and **z**), default value: 1000;
**pcg_maxit_u** : pcg maximum number of iterations allowed in inner loop (**u**), default value: 1000;
**prec** : preconditioner type for pcg when solving for **u** (D or BD), default value: D;
**verbosity** : if 1, prints external loop info, if 2 adds also **u**, **s**, **z** loops info, default value: 0.

TileCreator Tags:

**overlap** : number of pixels to be used as overlap between tiles, for each direction, use comma-separated values, default value: 0;
**tilesplit** : number of tiles, for each direction, use comma-separated values, default value: 0.

Main tags:

**printpars** : If greater than 0, prints all the parameters, default value: 0;
**outputFileTemplate** : template for output file names, default value: empty;
**initial_u** : file containing initial value for **u**, default value: empty;
**initial_s** : file containing initial value for **s**, default value: empty;
**initial_z** : file containing initial value for **z**, default value: empty.

Algorithms exit is reached when at least one of the stop criteria (relative decrease on function value, threshold on function value, maximum number of allowed iterations) is satisfied. OPARBCDAThreadManager_test will perform parallel tiled segmentation using OPARBCDA algorithm: in addition to OPARBCDAManager tags, it will accept also:

Parallelization Tags:

**workers** : number of workers, default value: 0.

# 4 MATLAB helper functions

## 4.1 Image file

In order to simplify the comparison with existing MATLAB code, the following are provided:

**readData** : reads a N-dimensional image from binary file;
**writeData** : writes a N-dimensional image to file.

Both functions handle little endian binary files containing:

- a signed 32 bit integer, representing the number of dimensions;
- for each dimension, a signed 32 bit integer, representing the number of elements along that dimension;

- a variable number of double precision floating point numbers.

Order of the dimension sizes match data layout: last size contains the number of contiguous elements in memory. As an example, given a $13x7$ random matrix:

```
A = rand(13,7);
writeData('A_bin', A);
```

the resulting `A_bin` file will contain:

- a signed 32 bit integer: 2;
- two signed 32 bit integers: 7 and 13;
- 91 double precision floating point numbers: $A(1,1)$, $A(2,1)$, ..., $A(13,1)$, $A(1,2)$, ..., $A(13,6)$, $A(13,7)$.

Images obtained by one of the executables can be read back in Matlab® with `readData.m`.

### 4.2 Parameters file

The following functions handle the creation/read of a binary file containing objective function parameters:

**readPars** : reads a file containing objective function parameters;
**writePars** : writes a file containing objective function parameters.

As an example, we can create a parameters file with teh following Matlab® lines:

```
% write objective function parameters to file
params.e  = 1e-2;
params.d  = 1;
params.a  = 2;
params.b  = 1;
params.mu = 0.5e-1;
params.tx = 1;
params.ty = 1;
params.xe = nthroot(st.e, 4);
params.oe = st.e^2;

writePars('my_parameters', params);
```

## 5 Tile generation and parallelization

Input image can be subdivided in different tiles: number of tiles for each dimension is set by using `tilesplit` tag. Overlap between consecutive tiles is controlled by `overlap` tag. Finally, by using `workers`, one can select the number of computational threads.

## 6 Examples

In this section, a small number of examples are shown in order to familiarize with the executables. Throughout the examples, we suppose to have input data in `CPP/data` and to collect ouput data in `CPP/outdata`; moreover we already compiled source files and we open a terminal in `CPP` folder.

### 6.1 Full image serial image segmentation

Let's select image contained in: `DATA_trento_g` and parameters in `DATA_trento_pars`: A full image (not tiled) segmentation process can be obtained using:

```
bintest/BCDAfullimage_test objbinfile=data/DATA_trento_pars data/
    DATA_trento_g
```

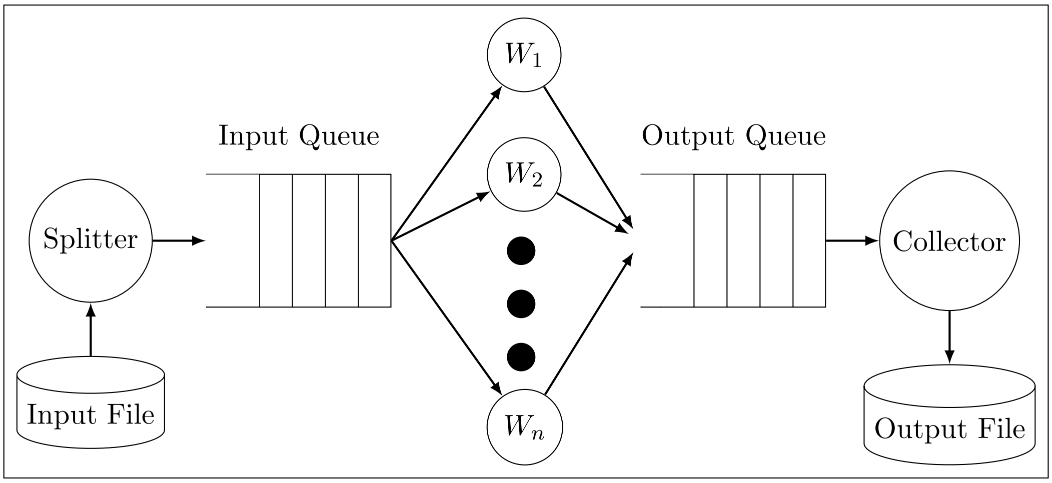


Figure 1: Tiled image segmentation approach represented as a queuing system.

This will launch BCDA segmentation algorithm: since no `outputFileTemplate` is provided, executable will create output file names starting with the name of the file used as input and with following trailing strings: `_u_sol`, `_s_sol` and `_z_sol`. In this example we then find **u** solution in `data/DATA_trento_g_u_sol`, **s** solution in `data/DATA_trento_g_s_sol` and **z** solution in `data/DATA_trento_g_z_sol`.

More details on the algorithm behaviour can be obtained by requesting to print all parameters (`printpars=1`), and to be verbose during the process (`verbose=1`); moreover we can specify output folder and prefix string for output files through the option `outputFileTemplate=outdata/DATA_trento`.

```
bintest/BCDAfullimage_test printpars=1 verbose=1 objbinfile=data/
    DATA_trento_pars outputFileTemplate=outdata/DATA_trento data/
    DATA_trento_g
```

[] Output files will be placed in `outdata` folder and named `DATA_trento_u_sol`, `DATA_trento_s_sol` and `DATA_trento_z_sol`. In the following we collected output of the executable:

```
Image dimension: 2020 2020

Objective Function Parameters:
 e  = 0.01
 d  = 30
 a  = 2
 b  = 1
 mu = 1
 tx = 1
 ty = 1
 xe = 0.316228
 oe = 0.0001

Solver Parameters:
 fval_tol     = 0.0005
 pcg_tol_sz   = 0
 pcg_tol_u    = 0
 ext_maxit    = 1000
 pcg_maxit_sz = 1000
 pcg_maxit_u  = 1000
 prec         = D
 verbosity    = 1

Overall Parameters:
```

```
25 printpars = 1
26 outputFileTemplate = outdata/DATA_trento
27 initial_u =
28 initial_s =
29 initial_z =
30
31 iter energy
32   0 1.500243e+12
33   1 1.103501e+08
34   2 9.461993e+07
35   3 9.126143e+07
36   4 8.996775e+07
37   5 8.921919e+07
38   6 8.873558e+07
39   7 8.839993e+07
40   8 8.815763e+07
41   9 8.797674e+07
42  10 8.783694e+07
43  11 8.772446e+07
44  12 8.763214e+07
45  13 8.755744e+07
46  14 8.749593e+07
47  15 8.744425e+07
48  16 8.739963e+07
49  17 8.736059e+07
50 Elapsed time 1.422827e+02 [s]
51 Exit reason: DESIRED TOLERANCE IS REACHED
52 Total iterations: 17 (s: 18) (z: 33) (u: 609)
53 Objective function at end: 8.736059e+07
```

Function value at line 53 can be used as a reference when calling OPARBCDA.

Linear algebra parallelization is obtained through OpenMP[4]: as every executable that exploits this technology, the number of threads is set through `OMP_NUM_THREADS` environmental variable, so for Bash shell you can use:

```
export OMP_NUM_THREADS=4
```

and, from now on, each OpenMP executable will be run using 4 threads. WARNING: since all executables provided in this package will read `OMP_NUM_THREADS`, it is good procedure to check it's value prior to launch a run:

```
echo $OMP_NUM_THREADS
```

### 6.2 Tiled serial image segmentation

Tiled segmentation process involving a $16 \times 16$ tile grid and overlap size of 10 for each direction can be obtained using:

```
bintest/OPARBCDAManager_test over_lb=8.736059e+07
 tilesplit=16,16 overlap=10,10 objbinfile=data/DATA_trento_pars
 outputFileTemplate=outdata/DATA_trento_ov data/DATA_trento_g
```

In this case, the algorithm is stopped when it reaches an objective function value lower than the lower bound specified on input. If we also add `printpars=1` and `verbosity=1` we obtain both the values of used parameters and some information concerning the stop criterion used for each internal subproblem.

End of output will read:

```
Elapsed time 3.432282e+02 [s]
Fini= 1.500243e+12
Fend= 8.708926e+07
Exit reason: DESIRED LOWER BOUND IS REACHED
Total iterations: 3 (s: 10842) (z: 12313) (u: 378581)
```

### 6.3 Tiled parallel image segmentation

Tiled segmentation process, using 4 workers, can be obtained using:

```
bintest/OPARBCDAThreadManager_test over_lb=8.736059e+07
 tilesplit=16,16 overlap=10,10 objbinfile=data/DATA_trento_pars
 outputFileTemplate=outdata/DATA_trento_ov workers=4 data/DATA_trento_g
```

End of output will read:

```
Elapsed time 1.695973e+02 [s]
Fini= 1.500243e+12
Fend= 8.708926e+07
Exit reason: DESIRED LOWER BOUND IS REACHED
Total iterations: 3 (s: 10842) (z: 12313) (u: 378582)
```

## References

## References

[1] Steven Dalton, Nathan Bell, Luke Olson, and Michael Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations. `http://cusplibrary.github.io`.

[2] Jared Hoberock and Nathan Bell. Thrust: A parallel template library. `https://thrust.github.io`.

[3] Dimitri van Heesch. `http://www.doxygen.org`.

[4] OpenMP Architecture Review Board. OpenMP application program interface. `http://www.openmp.org`.