FLCC Library version 1.5
User Manual

Georgios Papamakarios   Georgios Rizos

April 2013

Contents

Chapter 1

Introduction

The FLCC Library (which stands for Fast Local Correlation Coefficients) is a software tool that provides an interface for the fast computation of two fundamental image processing operations: the distribution of Correlation Coefficients with Local Normalization (also known as LCCs) and the sum of Convolution, between an image (or a stream of images) and an image template. Generally speaking, LCCs and Convolution are basic image-based information processing steps that find numerous applications in a wide spectrum of areas concerning image processing and computer vision, such as template or pattern matching, image registration, motion detection and many more. However, these operations (especially LCCs) have always been considered to be time-consuming and of high arithmetic complexity, particularly for real-time applications, thus making their usage rather troublesome.

This library intends to overcome this problem and provide users with a simple yet powerful interface for carrying out the computations under consideration. Especially in the case of LCCs, many implementations so far have tried to reduce computation time by sacrificing the local normalization characteristic of the LCCs or approximating the result in other lossy ways. This library, though, manages to reduce computation time to a minimum, yet without making any compromises on the quality of the result. The user can be sure that the output result is accurately the "real" LCC distribution in any case, exactly as it is defined formally.

The performance benefit is achieved by two routes. On the one hand, the library implements a set of optimized fast algorithms for computing LCCs or convolutions, selecting the most appropriate for each case, depending on the size of the input images and templates and the capability of the machine it runs on. On the other hand, it fully exploits current top-notch architectures, namely multicore CPU processors and manycore GPU devices. In other words, the library takes advantage of the system's computational resources, executing in parallel on multiple CPU threads or having the work load carried out by the powerful GPU devices of the system (according to what it deems to be faster).

The FLCC Library Version 1.5, although being fully operational, is still in a stage of infancy and more features are sure to be added in the future. For now, FLCC Library Version 1.5 features:

- Fast LCC computation of 2D and 3D images of any size
- Fast Convolution computation of 2D and 3D images of any size
- Both single and double-precision arithmetic
- Accelerated LCC or Convolution computation of a stream of images with the same template
- Ability to execute parallel computations on multiple CPUs or (currently one) GPU

This document presents the library interface. It first describes the data types and definitions used by the FLCC interface. Afterwards, the library routines and their usage are presented. Finally, it concludes with compiling and running instructions and a few code examples.

Chapter 2

Data Types

This section describes the data types and definitions used by the FLCC interface.

2.1 Type flccPlan

is an object which serves as a handle for FLCC execution configuration. It stores the plan containing the information by which the FLCC Library intends to execute the LCC or convolution computation for the specified parameters of a problem.

2.2 Type flccSize

```
typedef struct {
    int h;
    int w;
    int d;
} flccSize;
```

is a structure used for storing the size of the images which take part in the computation. The FLCC Library considers images to be arrays of real values, corresponding to image pixel values, while it currently supports 2D and 3D images. In the case of 2D images, variables h and w must be used to store the images' size and variable d can be ignored (otherwise the execution will fail). Note that the images must be stored as linear consecutive memory space in row-major order (also known as C-order). The dimensions' order is h, w for the 2D case and h, w, d for the 3D case. The row major format is described in further detail later in this document.

2.3 FLCC Computation Types

The FLCC Library supports both single image and streaming computation of LCCs and convolutions. That is, it can apply the template either on a single image (the simplest case) or a stream of consecutive images of the same size, one by one (especially useful when for example video is concerned). The significance of the second case, apart from being handy, is that execution time is considerably reduced compared to explicitly applying the same template on each image again and again.

Type flccType is an enumeration of two values used to define whether the type of planning that is to be done concerns one single computation between a single image and a template, or a series of computations between a stream of images and the same template.

```
typedef enum {
    FLCC_SINGLE,
    FLCC_STREAM
} flccType;
```

For example, if you intend to compute an LCC distribution between a single image and a template you should choose FLCC_SINGLE. On the other hand, if you have one template and a stream of images of the same size and you want the LCC distribution between each and every image and the template, you should choose FLCC_STREAM. Exactly the same applies for convolution computations, as the same computation types are used. This choice is to be made when planning, by use of functions lcorr_plan and conv_plan, as it is crucial concerning the validity of their output. The usage of the FLCC Computation Types is explained furthermore later in this document, below the sections of the planning functions.

2.4 FLCC Platforms

The FLCC Library enables the choice of the platform upon which the user wishes the computation to be carried out. Currently, the library can run either on the CPU (or CPUs) or, if it exists, on a single GPU device (later, we have in mind to implement the usage of multiple GPUs). The user can specify the platform to be used by the following enumeration of types:

```
typedef enum {
    FLCC_HOST,
    FLCC_DEVICE,
    FLCC_ANY
} flccPlatform;
```

where FLCC_ANY denotes any platform between the host and the device (meaning that the platform will be selected by the library, not the user).

This choice is stated only at the planning part of FLCC, whether it is for LCC or convolution computation. Depending on the choice, a certain subset of algorithms will be tested by the plan functions (lcorr_plan or conv_plan; this choice has the same gravity for both of these computations) and thus it will take less time to execute (with the exception of FLCC_ANY where all the algorithms will be tested). This may have an effect on the execution function (lcorr_exec or conv_exec) as maybe the plan will not be optimal. This choice exists for debugging reasons, or if the user is absolutely sure that the one platform will certainly be slower than the other and in which case does not want to wait longer for the plan function to complete. Of course, some users might not own a GPU, in which case it is a redundant choice to have the planner also try algorithms for a GPU. The most sensible choice however (if you do have a machine equipped with a GPU and waiting time is not such a big issue) is FLCC_ANY, as thus the plan will utilize the full potential of FLCC Library. Type flccPlatform is an enumeration of the aforementioned three values to be used in any way the user deems necessary.

2.5 Type flccResult

is an enumeration of values used as API function return values. The full set of defined return values and their meaning is as follows:

FLCC_SUCCESS	All FLCC operations are successful
FLCC_INVALID_PLAN	FLCC is passed an invalid plan handle
FLCC_INVALID_DIMENSION	The user specifies an unsupported dimension number
FLCC_INVALID_SIZE	The user specifies an unsupported size
FLCC_INVALID_TYPE	The user specifies a non-existent FLCC type
FLCC_INVALID_PLATFORM	The user specifies a non-existent platform
FLCC_INVALID_VALUE	The user specifies a bad memory pointer or an invalid array
FLCC_ALLOC_FAILED	FLCC failed to allocate memory
FLCC_EXEC_FAILED	FLCC failed to execute due to an internal error

Chapter 3

Functions

The FLCC Library is a lot similar to the FFTW and CUFFT libraries' plan/execute model. The plan is a configuration mechanism containing a set of information describing the optimal execution of a particular problem.

The FLCC interface includes 6 sets of main functions, 3 for LCC computation and 3 for convolution computation. Each set consists of a single and a double-precision version of a particular function. There are also 2 additional functions for memory allocation and deallocation. All of them are described in detail below.

3.1 lcorr_plan_f / lcorr_plan_d

```
flccResult lcorr_plan_f(flccPlan *plan, int dim,
    flccSize imSize, flccSize temSize, flccType type,
    flccPlatform platform)

flccResult lcorr_plan_d(flccPlan *plan, int dim,
    flccSize imSize, flccSize temSize, flccType type,
    flccPlatform platform)
```

Function lcorr_plan_X creates the LCC plan for the specified execution configuration. "X" is either "f" or "d", indicating respectively the single or double-precision version of this type of function. Even though no information regarding the precision is dictated by the arguments, the appropriate function must be chosen in order to obtain the optimal result FLCC may achieve. It measures execution times of a number of different LCC algorithms and selects the optimal one for the machine it runs on. The information generated during lcorr_plan_X execution is stored in a type flccPlan object. This plan object can later be used for multiple LCC computation executions of different arrays of the same configuration. In fact, this is the inherent power of the plan/execute model, that the planning is done only once, while the execution can be done an infinite number of times, thus making the planning cost insignificant and improving the overall performance significantly.

The size of both the image and the template are needed by this function, which together determine the size of the problem, as well as the dimension of the arrays taking part in the computation. Note that the arrays must be 2 or 3 dimensional and their sizes on each dimension must be greater than 1. That means that the user can't configure a 2D computation by stating a 3D one and declaring one size as 1.

The plan may be used for a single LCC computation between a single image and a template, or a series of computations between a stream of images of the same size and the same template. Notice that in the case of a stream of images, the size of one image must be given, as if it would be a single image. The number of images for the stream and the actual array complete with its values are arguments for lcorr_exec_X (explained later). You have to be careful in choosing the type parameter here, and use the plan only for the purpose intended (single or streaming).

If you don't have restrictions on the platform and wish FLCC Library to test the full set of algorithms (both those that execute on the CPU host and those that execute on the GPU device), you should choose as a value for the parameter platform FLCC_ANY. Also, if the greater execution time of lcorr_plan_X is not a grave problem we recommend FLCC_ANY. This way, FLCC Library is used optimally, meaning it selects the most "efficient" between the CPU and the GPU. If though you do want an LCC computation on a certain platform, or maybe you do not own a GPU, you should consider the other choices as they lessen the execution time of this function. Finally, if you have had FLCC installed without its CUDA component, selecting FLCC_DEVICE will cause lcorr_plan_X to fail.

It should be noted that, for backwards compatibility reasons, there is included in FLCC v1.5 the function lcorr_plan that works exactly like lcorr_plan_f.

INPUT	
plan	Pointer to an flccPlan object
dim	Number of dimensions (2 or 3)
imSize	Number of floats/doubles constituting one image
temSize	Number of floats/doubles constituting the template
type	Single or streaming LCC computation (e.g. FLCC_SINGLE or FLCC_STREAM)

platform	Test algorithms for CPUs, GPU or both (e.g. FLCC_HOST, FLCC_DEVICE or FLCC_ANY)

OUTPUT	
plan	Contains an flccPlan handle value

RETURN VALUES	
FLCC_SUCCESS	FLCC successfully created the plan
FLCC_INVALID_PLAN	FLCC is passed an invalid plan handle
FLCC_INVALID_DIMENSION	FLCC supports only 2D or 3D problems
FLCC_INVALID_SIZE	The sizes parameters are unsupported
FLCC_INVALID_TYPE	Parameter type is non-existent
FLCC_INVALID_PLATFORM	Parameter platform is non-existent
FLCC_ALLOC_FAILED	FLCC failed to allocate memory
FLCC_EXEC_FAILED	FLCC failed to execute due to an internal error

3.2 conv_plan_f / conv_plan_d

```
flccResult conv_plan_f(flccPlan *plan, int dim,
    flccSize imSize, flccSize temSize, flccType type,
    flccPlatform platform)

flccResult conv_plan_d(flccPlan *plan, int dim,
    flccSize imSize, flccSize temSize, flccType type,
    flccPlatform platform)
```

Function conv_plan_X works exactly in the same way as lcorr_plan_X, measuring execution time of different convolution algorithms. “X” is either “f” or “d”, indicating respectively the single or double-precision version of this type of function. Even though no information regarding the precision is dictated by the arguments, the appropriate function must be chosen in order to obtain the optimal result FLCC may achieve. The information generated during conv_plan_X execution is again stored in a type flccPlan object. This plan object can later be used for multiple convolution computation executions of different arrays of the same configuration.

The size of both the image and the template are needed by this function, which together determine the size of the problem, as well as the dimension of the arrays taking part in the computation. Note that the arrays must be 2 or 3 dimensional and their sizes on each dimension must be greater than 1. That means that the user can't configure a 2D computation by stating a 3D one and declaring one size as 1.

The plan may be used for a single convolution computation between a single image and a template, or a series of computations between a stream of images of the same size and the same template. Notice that in the case of a stream of images, the size of one image must be given, as if it would be a single image. The number of images for the stream and the actual array complete with its values are arguments for conv_exec_X (explained later). You have to be careful in choosing the type parameter here, and use the plan only for the purpose intended (single or streaming).

If you don't have restrictions on the platform and wish FLCC Library to test the full set of algorithms (both those that execute on the CPU host and those that execute on the GPU device), you should choose as a value for the parameter platform FLCC_ANY. Also, if the greater execution time of conv_plan_X is not a grave problem, we recommend FLCC_ANY. This way, FLCC Library is used optimally, meaning it selects the most "efficient" between the CPU and the GPU. If though you do want a convolution computation on a certain platform, or maybe you do not own a GPU, you should consider the other choices as they lessen the execution time of this function. Finally, if you have had FLCC installed without its CUDA component, selecting FLCC_DEVICE will cause conv_plan_X to fail.

It should be noted that, for backwards compatibility reasons, there is included in FLCC v1.5 the function conv_plan that works exactly like conv_plan_f.

INPUT	
plan	Pointer to an flccPlan object
dim	Number of dimensions (2 or 3)
imSize	Number of floats/doubles constituting one image
temSize	Number of floats/doubles constituting the template
type	Single or streaming convolution computation (e.g. FLCC_SINGLE or FLCC_STREAM)
platform	Test algorithms for CPUs, GPU or both (e.g. FLCC_HOST, FLCC_DEVICE or FLCC_ANY)

OUTPUT	
plan	Contains an fllccPlan handle value

RETURN VALUES	
FLCC_SUCCESS	FLCC successfully created the plan
FLCC_INVALID_PLAN	FLCC is passed an invalid plan handle
FLCC_INVALID_DIMENSION	FLCC supports only 2D or 3D problems
FLCC_INVALID_SIZE	The sizes parameters are unsupported
FLCC_INVALID_TYPE	Parameter type is non-existent
FLCC_INVALID_PLATFORM	Parameter platform is non-existent
FLCC_ALLOC_FAILED	FLCC failed to allocate memory
FLCC_EXEC_FAILED	FLCC failed to execute due to an internal error

3.3 lcorr_exec_f / lcorr_exec_d

```
flccResult lcorr_exec_f(flccPlan plan, float *image,
    float *templat, float *lcc, int imCount)

flccResult lcorr_exec_d(flccPlan plan, double *image,
    double *templat, double *lcc, int imCount)
```

Function lcorr_exec_X executes an LCC computation according to the information stored in a type flccPlan object, which has already been created by function lcorr_plan_X. “X” is either “f” or “d”, indicating respectively the single or double-precision version of this type of function.

Apart from the plan, needed are also: a pointer to the float/double array of the image and a pointer to the float/double array of the template. The template must be of template size as stated during plan configuration. The matter of the size of the image is a little more complicated. In case the user wants a single LCC distribution of a single image (in other words the plan was configured using FLCC_SINGLE), the array should be of image size (as decreed during the call of lcorr_plan_X). In case of a stream of images, the array must contain the first image, immediately afterwards the second et cetera. The size will be the image size multiplied by imCount. It should be highlighted that the type of the computation (single or streaming) is determined by the plan and not by this function. It is the responsibility of the user to use the right plan for the right purpose.

Moreover, in the case of streaming computation, the number of images to be processed must be declared beforehand as well, in the form of parameter imCount. In the case of a single LCC computation, the parameter imCount is ignored and only one image is processed (that means that imCount can be anything—we recommend the value 1 to be used for clarity reasons). Also, if you give a pointer to an array which contains more images than you declare, lcorr_exec_X will NOT process them all—just the first n the user declares. Naturally FLCC may process garbage if you give an imCount greater than the number of images to be processed, so the user must be extra careful when calling lcorr_exec_X.

The elements in both cases are to be given in row major format where the dimensions are in the following order: h, w, d. That means that dimension-wise the index which changes the most frequently is d, then w, then h. Actually, this means that you have to be careful to make the according declaration in the type flccSize objects imSize and temSize.

The array lcc in which the LCC distribution is to be stored must be preallocated by the user. Again, in the case of a single LCC computation the array must be of convolution size (explained later), while in the case of a stream of images lcc array must be of convolution size multiplied by imCount. This function, fills the lcc array with the local correlation coefficients in row major order as explained before.

Convolution size:

Let $ih \times iw \times id$ be the image size where ih, iw and id are the sizes of the array along each dimension. The same with th, tw and td for the template. Let sh, sw and sd be the sizes of lcc array. Then:

$$sh = ih + th - 1, \quad sw = iw + tw - 1, \quad sd = id + td - 1$$

Thus, convolution size = $sh \times sw \times sd$. (In the calculations above ignore dimension d in the 2D case.)

It should be noted that, for backwards compatibility reasons, there is included in FLCC v1.5 the function lcorr_exec that works exactly like lcorr_exec_f.

INPUT	
plan	Contains an flccPlan handle value
image	Pointer to a row major float/double array or a stream of row major float arrays
templat	Pointer to a row major float/double array
lcc	Pointer to a float/double array
imCount	The number of images to be processed

OUTPUT	
lcc	LCC between image or stream of images and template

RETURN VALUES	
FLCC_SUCCESS	FLCC successfully executed an LCC computation
FLCC_INVALID_PLAN	FLCC is passed an invalid plan handle
FLCC_INVALID_SIZE	Parameter imCount is less than 1 in case of streaming (otherwise it can be anything)
FLCC_INVALID_VALUE	Parameters image, templat or lcc are invalid
FLCC_ALLOC_FAILED	FLCC failed to allocate memory
FLCC_EXEC_FAILED	FLCC failed to execute due to an internal error

3.4 conv_exec_f / conv_exec_d

```
flccResult conv_exec_f(flccPlan plan, float *image,
    float *templat, float *conv, int imCount)

flccResult conv_exec_d(flccPlan plan, double *image,
    double *templat, double *conv, int imCount)
```

Function conv_exec_X works exactly like lcorr_exec_X in that it executes a convolution computation according to the information stored in a type flccPlan object, which has already been created by function conv_plan_X. "X" is either "f" or "d", indicating respectively the single or double-precision version of this type of function.

Apart from the plan, needed are also: a pointer to the float/double array of the image and a pointer to the float/double array of the template. The template must be of template size as stated during plan configuration. The matter of the size of the image is a little more complicated. In case the user wants a single convolution of a single image (in other words the plan was configured using FLCC_SINGLE), the array should be of image size (as decreed during the call of conv_plan_X). In case of a stream of images, the array must contain the first image, immediately afterwards the second et cetera. The size will be the image size multiplied by imCount. It should be highlighted that the type of the computation (single or streaming) is determined by the plan and not by this function. It is the responsibility of the user to use the right plan for the right purpose.

Moreover, in the case of streaming computation, the number of images to be processed must be declared beforehand as well, in the form of parameter imCount. In the case of a single convolution computation, the parameter imCount is ignored and only one image is processed (that means that imCount can be anything—we recommend the value 1 to be used for clarity reasons). Also, if you give a pointer to an array which contains more images than you declare, conv_exec_X will NOT process them all—just the first n the user declares. Naturally, FLCC may process garbage if you give an imCount greater than the number of images to be processed, so the user must be extra careful when calling conv_exec_X.

The elements in both cases are to be given in row major format where the dimensions are in the following order: h, w, d. That means that dimension-wise the index which changes the most frequently is d, then w, then h. Actually, this means that you have to be careful to make the according declaration in the type flccSize objects imSize and temSize.

The array conv in which the convolution is to be stored must be preallocated by the user. Again, in the case of a single convolution computation the array must be of convolution size (explained later), while in the case of a stream of images conv array must be of convolution size multiplied by imCount. This function, fills the conv array with the convolution in row major order as explained before.

Convolution size:

Let $ih \times iw \times id$ be the image size where ih, iw and id are the sizes of the array along each dimension. The same with th, tw and td for the template. Let sh, sw and sd be the sizes of conv array. Then:

$$sh = ih + th - 1, \quad sw = iw + tw - 1, \quad sd = id + td - 1$$

Thus, convolution size $= sh \times sw \times sd$. (In the calculations above ignore dimension d in the 2D case.)

It should be noted that, for backwards compatibility reasons, there is included in FLCC v1.5 the function conv_exec that works exactly like conv_exec_f.

INPUT	
plan	Contains an flccPlan handle value
image	Pointer to a row major float/double array or a stream of row major float arrays
templat	Pointer to a row major float/double array
conv	Pointer to a float/double array

imCount	The number of images to be processed

OUTPUT	
conv	Convolution between image or stream of images and template

RETURN VALUES	
FLCC_SUCCESS	FLCC successfully executed an convolution computation
FLCC_INVALID_PLAN	FLCC is passed an invalid plan handle
FLCC_INVALID_SIZE	Parameter imCount is less than 1 in case of streaming (otherwise it can be anything)
FLCC_INVALID_VALUE	Parameters image, templat or conv are invalid
FLCC_ALLOC_FAILED	FLCC failed to allocate memory
FLCC_EXEC_FAILED	FLCC failed to execute due to an internal error

3.5 lcorr_destroy_f / lcorr_destroy_d

```
flccResult lcorr_destroy_f(flccPlan *plan)

flccResult lcorr_destroy_d(flccPlan *plan)
```

Function lcorr_destroy_X frees all resources occupied by an flccPlan object created by lcorr_plan_X, thus making it unusable. “X” is either “f” or “d”, indicating respectively the single or double-precision version of this type of function. It should be called if the plan is no more needed.

It should be noted that, for backwards compatibility reasons, there is included in FLCC v1.5 the function lcorr_destroy that works exactly like lcorr_destroy_f.

INPUT	
plan	Points to an flccPlan object

RETURN VALUES	
FLCC_SUCCESS	FLCC successfully destroyed the plan
FLCC_INVALID_VALUE	Bad memory pointer
FLCC_EXEC_FAILED	FLCC failed to destroy the plan due to an internal error

3.6 conv_destroy_f / conv_destroy_d

```
flccResult conv_destroy_f(flccPlan *plan)

flccResult conv_destroy_d(flccPlan *plan)
```

Function conv_destroy_X frees all resources occupied by an flccPlan object created by conv_plan_X, thus making it unusable. “X” is either “f” or “d”, indicating respectively the single or double-precision version of this type of function. It should be called if the plan is no more needed.

It should be noted that, for backwards compatibility reasons, there is included in FLCC v1.5 the function conv_destroy that works exactly like conv_destroy_f.

INPUT	
plan	Points to an flccPlan object

RETURN VALUES	
FLCC_SUCCESS	FLCC successfully destroyed the plan
FLCC_INVALID_VALUE	Bad memory pointer
FLCC_EXEC_FAILED	FLCC failed to destroy the plan due to an internal error

3.7 flcc_malloc

```
void *flcc_malloc(size_t bytes)
```

All the arrays that are passed as input arguments in lcorr_exec_X and conv_exec_X, namely the image (or stream of images), the template and the output (LCC or convolution), must have already been allocated in page-locked

CPU memory. To facilitate this process and to relieve the user of the details of it, we provide this function for memory allocation, which works similarly to malloc and makes sure the aforementioned conditions are met. Thus, the discussed arrays must always be allocated by this function. Note that if usual malloc or new are used instead, the FLCC execution routines will probably fail.

INPUT	
bytes	Number of bytes to be allocated

OUTPUT

A void pointer pointing to the first position of the allocated memory. It should be typecasted to the needed type (here, float * or double *). If allocation fails, NULL is returned.

3.8 flcc_free

```
void flcc_free(void *array)
```

This function frees the memory previously allocated by flcc_malloc. It works similarly to free, and it should be used when the array is no longer needed. Note that memory allocated by flcc_malloc cannot be deallocated by free, so this function must be used instead.

INPUT	
array	The array to be deallocated

Chapter 4

Matlab Interface

Additionally to the C interface described above, there is also the possibility to use FLCC directly from Matlab. For every C function of FLCC there exists a corresponding Matlab function of the same name, which invokes the former internally and therefore performs the same action without the user having to leave the Matlab environment. This way, FLCC's Matlab interface manages to combine the efficiency and high performance of FLCC with the comfort and straightforwardness of the Matlab programming environment. In this chapter we describe in detail the calling mode of the FLCC's Matlab interface functions.

4.1 lcorr_plan and conv_plan

```
plan = lcorr_plan(imSize, temSize, type, platform,
                  precision)

plan = conv_plan(imSize, temSize, type, platform,
                 precision)
```

These are the planner functions of FLCC as they appear in the Matlab programming environment, for LCC and convolution respectively. They work exactly as the C functions of the same name and they produce the optimal plan for each computation specification.

INPUT	
imSize	One-dimensional Matlab array describing the size of the image; may have 2 or 3 elements for a 2D or a 3D image respectively

temSize	One-dimensional Matlab array describing the size of the template; may have 2 or 3 elements for a 2D or a 3D template respectively
type	Matlab string specifying the computation type (can be ‘FLCC_SINGLE’ or ‘FLCC_STREAM’)
platform	Matlab string specifying the computation platform (can be ‘FLCC_HOST’, ‘FLCC_DEVICE’ or ‘FLCC_ANY’)
precision	Matlab string specifying the desired precision (can be ‘single’ or ‘double’)

OUTPUT	
plan	Contains a plan handle value for FLCC

Note that the arguments for the type, the platform and the precision are optional; in case they are omitted, their default values become FLCC_SINGLE, FLCC_ANY and single respectively.

4.2 lcorr_exec and conv_exec

```
lcc = lcorr_exec(plan, image, templat)

conv = conv_exec(plan, image, templat)
```

These are the Matlab analogues for the FLCC execution functions. They perform LCC and convolution respectively according to the specified plan. The latter has to be the output of the corresponding Matlab planner function.

There is an important distinction in the way the above functions are called, depending on whether we have a single image computation or streaming. In the case of a single image, the image is passed to the function in the form of a 2D or 3D Matlab array and the result is returned as a 2D or 3D Matlab array as well. In the case of streaming, though, the whole image stream needs to be passed to the function and it may consist of one or more images. In this case the user needs to pass the images as a cell array, with each cell containing a different image in the form of a 2D or 3D Matlab array. The output is also returned as a cell array of the same size as the input image cell array, with each cell containing the result for the corresponding image. There are no restrictions on the size and dimensionality of the cell array. The only restriction regards the size of the images; they all need to be of the same size for the streaming computation to be meaningful. In any other case, the result is undefined.

The image and template to be passed as arguments may either be of single or double-precision (with the result being of single or double-precision respectively). It is the user's responsibility to make sure that the plan has been generated using the appropriate precision specifier in the planner function.

INPUT	
plan	The plan handle returned by a call to the corresponding planner function
image	Matlab array describing the image; cell array of images in the case of streaming
templat	Matlab array describing the template

OUTPUT	
lcc/conv	Matlab array describing the result of the computation; cell array in the case of streaming

4.3 lcorr_destroy and conv_destroy

```
lcorr_destroy(plan, precision)

conv_destroy(plan, precision)
```

Similarly to the C interface, the plan uses some memory resources that should be released after it is no longer needed. This can be done via the above functions. After its destruction, the plan is rendered unusable. Note that the precision specifier should be the same as in the plan creation. The argument is optional and defaults to single.

INPUT	
plan	The plan handle returned by a call to the corresponding planner function
precision	Matlab string specifying the precision the plan has been generated with (can be 'single' or 'double'; default is 'single')

Chapter 5

Install, Compile and Run

5.1 Installation Instructions

FLCC comes as a source code package and installs under UNIX/Linux systems. It can also be installed under Windows via Cygwin. Installing can be done simply by issuing “make install” in the package’s main folder. This will generate the library file, libflcc.a, in the folder lib under the package’s main folder, which then the user may link with their own programs. Moreover, to install the Matlab interface, issue “make matlab” in the package’s main folder. This will produce the executable mex files that implement FLCC’s functions in folder mex/bin.

In order to obtain the power to use the GPU devices, FLCC uses the CUDA programming model and architecture. This means that, in order to use FLCC’s GPU capability, the user must have CUDA (any version) already installed on their system and the GPUs to be used must be CUDA compatible.

Also, in order to obtain its full functionality, FLCC uses a number of libraries itself. The complete list is:

- Pthreads (any version)
- FFTW version 3.x with threads (both the single and double-precision versions)
- CUFFT (any version)

That practically means that the aforementioned libraries must be already installed on the system in order for FLCC to install successfully and to be fully functional, as described in this manual.

Nevertheless, in case the user doesn’t have CUDA and CUFFT installed on their system or if for any reason wishes so, FLCC can still be installed without them and operate only on the system's CPUs. To do so, the user will only need to uncomment a specific line in the makefile before installing. We should mention that in case FLCC is installed without CUDA and CUFFT, passing FLCC_DEVICE as a platform designator to the planning functions will make these functions fail. Apart from that, FLCC will work normally, except that it won't take advantage anymore of the GPU-based methods.

The same as above works for FFTW. In case the user doesn't have it, doesn't want to install it or for any reason wishes to do so, FLCC can be installed without FFTW by simply uncommenting a line in the makefile prior to installing. This won't change at all the way FLCC is used and FLCC will continue to perform just by ignoring the FFTW-based methods. We should note that installing without CUDA and installing without FFTW are two totally independent actions and thus the one can be done with or without the other.

In any case, although the user is free to do whatever they deem preferable, we suggest that if FFTW and/or CUDA are present it would be beneficial to be used, in order to get the full potential out of FLCC. Further details on installation issues can be found in the README file that comes with the package and in the makefile comments.

Since version 1.5, FLCC routines come in two versions, i.e. single and double-precision versions. During installation, both versions are installed by default. Nevertheless, by simply uncommenting a line in the makefile, it is possible that only the single or double-precision version be installed. Refer to the README file and the makefile's comments for further information.

A final issue on installation concerns the Matlab interface. In order for the executable mex files to be produced, the code to be linked to them must be position independent. Static libraries such as FFTW may therefore fail to link, causing installation to fail as well. If that is the case, the user needs to recompile those libraries, making them position independent. This is done with the flag -fPIC in the compiler. Refer to Matlab mex documentation for further details on the subject.

5.2 Compiling Instructions

The FLCC Library is an API on C (or C++) language, so the source files using it should be of .c or .cpp format and should be compiled by a suitable C or C++ compiler (for example gcc or g++). Also, they can be of .cu format and can be compiled by nvcc (which are the CUDA format and compiler respectively). Note that since FLCC uses the CUFFT library (which is a CUDA library), if you don't compile with nvcc then you will have to inform the compiler about the location of CUFFT, except of course if you have chosen to install without it.

We here remind that, in its full form, FLCC uses the following libraries:

- Pthreads (any version)
- FFTW version 3.x with threads (both the single and double-precision versions)
- CUFFT (any version)

Thus, during compiling your programs, linking these libraries is necessary (or at least those which have been included during installation). You don't need though to include header files of these libraries in your code. Linking them during compiling would be enough.

Finally, to compile with the FLCC Library you need to:

- Include in your code the file flcc.h (it can be found in folder inc under FLCC package's main folder)
- When compiling, link with -lflcc (in order to link with the file libflcc.a situated in folder lib of FLCC package's main folder)

Be sure to state the include path to the header file (flcc.h) and the library path to the library file (libflcc.a). Then you will be ready to compile, which in a Linux system is done as follows (in this example with the use of gcc compiler):

```
gcc foo.c -o foo -lflcc -lpthread -lfftw3f -lfftw3
    -lfftw3f_threads -lfftw3_threads -lm -lcufft
```

where foo.c is the source file that uses the FLCC Library and foo is the produced executable file. The paths to the header file and the library file could have also been stated here, by the use of flags -I and -L respectively, immediately followed by the according paths. Needless to say that if FLCC has been installed without FFTW and/or CUDA, the corresponding library should be missing from the above command. The same holds if only the single or double-precision version is installed; in that case, only FFTW's corresponding version (single or double-precision) is needed.

Finally, using FLCC's Matlab interface in your Matlab scripts is a rather straightforward task. After installation, folder mex/bin under the package's main folder will contain the executable mex files implementing FLCC's Matlab functions. As long as Matlab's path points to them, FLCC's Matlab functions can be used directly from your scripts. Refer to Matlab documentation on how to run Matlab programs for further information.

5.3 Running Issues

When running programs with FLCC Library, one should have in mind the following:

- FLCC does not directly support the computation of correlation (cross- or auto-). This was done on purpose, since convolution can be used in this case instead, by just rearranging the template array. More specifically, executing convolution with a template that is reversed along every dimension is identical to executing correlation. So the user can effectively use the convolution functions for performing correlations, passing as an argument the reversed template instead. In the same manner, passing a reversed template as an input to the LCC computation routines would produce locally normalized convolution.

- When the library executes on a GPU device, it doesn't select by itself the optimal device (in case more than one devices are existent and supported), but uses whichever device was previously set by the program. If no device had been explicitly selected before, the library selects the one labeled as 0. So, for maximum performance, the user should set the fastest device to be used before calling any library functions. This is to be concerned only if the system includes more than one devices. If only one GPU exists, then it will be selected automatically by the library routines and no specific action is required by the user.

- As it has already been mentioned, FLCC uses the FFTW Library. Thus, the user can accelerate the time performance of the planning functions by taking advantage of the wisdom feature of FFTW. To be more precise, FFTW accumulates information as it executes and that information (called wisdom) is used over and over again, increasing planning time performance as the program proceeds. The user can also save and import wisdom from previous programs. So, as FFTW accumulates wisdom and becomes faster, FLCC becomes faster too. To conclude, if you use FFTW independently, we strongly recommend to consider reusing the wisdom accumulated, since it will increase planning performance significantly. See FFTW documentation for further detail of this subject. Of course, this is not to be taken into consideration when FLCC is installed without FFTW.

Chapter 6

Example Code

6.1 Example 1

The following example is an LCC computation of a single 2D image of single-precision using the FLCC Library. In the planning part, only the algorithms that use a GPU are tested.

```
#include <flcc.h>

flccSize imSize, temSize;
int I_SIZE, T_SIZE, S_SIZE;

imSize.h  = 1000;
imSize.w  = 1500;
temSize.h = 50;
temSize.w = 40;

I_SIZE = imSize.h  * imSize.w;
T_SIZE = temSize.h * temSize.w;
S_SIZE = (imSize.h+temSize.h-1)*(imSize.w+temSize.w-1);

flccPlan plan;
flccResult r;

r = lcorr_plan_f (&plan, 2, imSize, temSize,
                  FLCC_SINGLE, FLCC_DEVICE);

if (r != FLCC_SUCCESS)
    exit(1);
```

```
//..............................................

float *image;
float *templat;
float *lcc;

image   = (float *)flcc_malloc( sizeof(float)*I_SIZE );
templat = (float *)flcc_malloc( sizeof(float)*T_SIZE );
lcc     = (float *)flcc_malloc( sizeof(float)*S_SIZE );

r = lcorr_exec_f (plan, image, templat, lcc, 1);

if (r != FLCC_SUCCESS)
    exit(1);

//..............................................

r = lcorr_destroy_f (&plan);

if (r != FLCC_SUCCESS)
    exit(1);

flcc_free (image);
flcc_free (templat);
flcc_free (lcc);
```

6.2 Example 2

The following example is the LCC computation of a stream of 20 3D images of double-precision using the FLCC Library. In the planning part, all the available algorithms are tested.

```
#include <flcc.h>

flccSize imSize, temSize;
int I_SIZE, T_SIZE, S_SIZE;

imSize.h  = 100;
imSize.w  = 150;
imSize.d  = 200;
temSize.h = 10;
temSize.w = 15;
temSize.d = 10;
```

```
I_SIZE = imSize.h  * imSize.w  * imSize.d;
T_SIZE = temSize.h * temSize.w * temSize.d;
S_SIZE = (imSize.h+temSize.h-1)*(imSize.w+temSize.w-1)*
         (imSize.d+temSize.d-1);

flccPlan plan;
flccResult r;

r = lcorr_plan_d (&plan, 3, imSize, temSize,
                  FLCC_STREAM, FLCC_ANY);

if (r != FLCC_SUCCESS)
    exit(1);

//..............................................

double *image;
double *templat;
double *lcc;

image  =(double*)flcc_malloc(sizeof(double)*I_SIZE*20);
templat=(double*)flcc_malloc(sizeof(double)*T_SIZE);
lcc    =(double*)flcc_malloc(sizeof(double)*S_SIZE*20);

r = lcorr_exec_d (plan, image, templat, lcc, 20);

if (r != FLCC_SUCCESS)
    exit(1);

//..............................................

r = lcorr_destroy_d (&plan);

if (r != FLCC_SUCCESS)
    exit(1);

flcc_free (image);
flcc_free (templat);
flcc_free (lcc);
```

6.3 Example 3

The following example is the convolution computation of a single 3D image of single-precision using the FLCC Library. In the planning part, only the algorithms that run on the CPU (or CPUs) are tested.

```
#include <flcc.h>

flccSize imSize, temSize;
int I_SIZE, T_SIZE, S_SIZE;

imSize.h  = 150;
imSize.w  = 200;
imSize.d  = 50;
temSize.h = 12;
temSize.w = 10;
temSize.d = 4;

I_SIZE = imSize.h  * imSize.w  * imSize.d;
T_SIZE = temSize.h * temSize.w * temSize.d;
S_SIZE = (imSize.h+temSize.h-1)*(imSize.w+temSize.w-1)*
         (imSize.d+temSize.d-1);

flccPlan plan;
flccResult r;

r = conv_plan_f (&plan, 3, imSize, temSize,
                 FLCC_SINGLE, FLCC_HOST);

if (r != FLCC_SUCCESS)
    exit(1);

//..................................................

float *image;
float *templat;
float *conv;

image   = (float *)flcc_malloc( sizeof(float)*I_SIZE );
templat = (float *)flcc_malloc( sizeof(float)*T_SIZE );
conv    = (float *)flcc_malloc( sizeof(float)*S_SIZE );

r = conv_exec_f (plan, image, templat, conv, 1);

if (r != FLCC_SUCCESS)
    exit(1);

//..................................................

r = conv_destroy_f (&plan);
```

```
if (r != FLCC_SUCCESS)
    exit(1);

flcc_free (image);
flcc_free (templat);
flcc_free (conv);
```

6.4 Example 4

Here we present a simple example of the Matlab interface of FLCC. The following could be part of a Matlab m-file. It computes the convolution between a 2D image and a template of single-precision. Note that the planning is implicitly done with FLCC_SINGLE and FLCC_ANY.

```
img = single(rand(1000,500));
tem = single(rand(16,16));

plan = conv_plan(size(img), size(tem));
con  = conv_exec(plan, img, tem);
conv_destroy(plan);
```

6.5 Example 5

Here we present an example of the Matlab interface of FLCC. The following could be part of a Matlab m-file. It computes the LCC between a stream of 10 3D images and the same template, all being of double-precision. It demonstrates the usage of cell arrays to denote streaming.

```
imgSize = [100 50 100];
temSize = [8 8 8];
imCount = 10;

img = cell(1,imCount);
for i=1:imCount
    img{i} = rand(imgSize);
end
tem = rand(temSize);

plan = lcorr_plan(imgSize, temSize, ...
    'FLCC_STREAM', 'FLCC_DEVICE', 'double');
lcc  = lcorr_exec(plan, img, tem);
```

```
lcorr_destroy(plan, 'double');
```