PFAC Library

GPU-based string matching algorithm

Authors
Chen-Hsiung Liu, lgen7604@gmail.com
Lung-Sheng Chien, lungshengchien@gmail.com
Cheng-Hung Lin, brucelinco@gmail.com
Shih-Chieh Chang, shihchiehchang@gmail.com

---

v1.0
February, 2011

Table of Contents

1. The PFAC library

PFAC is abbreviated from the **P**arallel **F**ailureless **A**ho-**C**orasick algorithm proposed in [1]. It is a variant of the well-known Aho-Corasick (AC) algorithm with all its failure transitions removed as well as the self-loop transition of the initial state. The purpose of PFAC is to match all longest patterns in a given input stream against patterns pre-defined by users.

1.1 Introduction of the PFAC algorithm

Using the PFAC algorithm for string matching invokes two steps. The first step is to construct a PFAC state machine.

For example: consider to match four patterns, AB, ABG, BEDE and ED. PFAC first transforms these four patterns into a PFAC state machine as shown in Figure 1. The PFAC state machine has 11 states labeled as 1, 1, ..., 10 where states 1, 2, 3 and 4 are final states. Each final state corresponds to a pattern. For example, state 1 corresponds to pattern 1 ("AB"), state 2 corresponds to pattern 2 ("ABG"), state 3 corresponds to pattern 3 ("BEDE") and state 4 corresponds to pattern 4 ("ED").

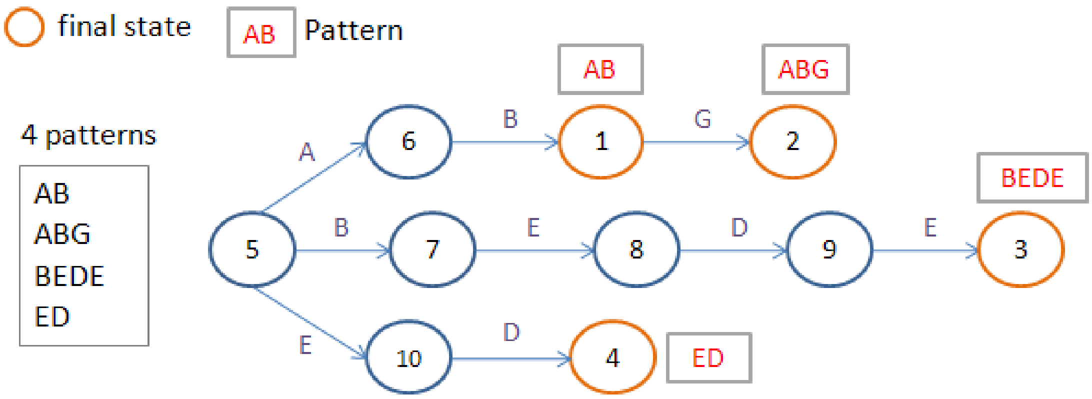

Figure 1: finite state machine of patterns {AB, ABG, BEDE, ED}

In the second step, the PFAC algorithm creates an individual thread for each byte of an input stream to identify any pattern starting at the thread's starting position. As shown in Figure 2, the number of threads created by the PFAC algorithm is equal to the length of an input stream.

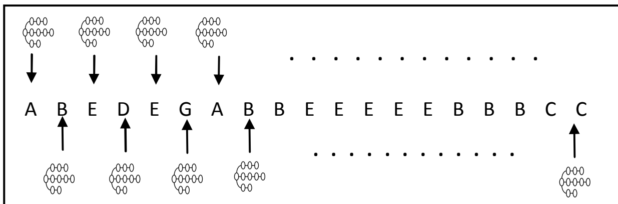

Figure 2. Parallel Failureless-AC algorithm

For example, suppose an input stream contains "ABEDEDABG", then PFAC invokes nine threads with thread ID, t_0, t_1, ... , t_8 to process this stream. Thread t_j checks a substring starting at position j. For example, t_1 checks the substring "BEDE..." and find a match of pattern 3 ("BEDE"). Every thread starts from initial state (state 4). In PFAC, a thread terminates if its next state is a trap state which indicates there is no valid transition for the current state and input character.

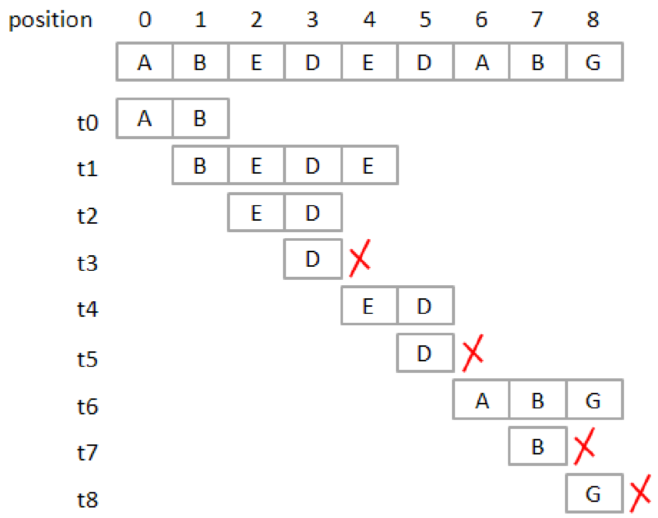

Figure 3: nine threads process input stream "ABEDEDABG" in parallel.

The idea of allocating an individual thread to each byte of an input stream has an important implication on the efficiency of the PFAC state machine where each thread of PFAC is only responsible for identifying the pattern starting at the thread starting position, it terminates immediately. For example, in Figure 3, thread t3 takes character 'D' in the initial state. Because there exists no valid transition for 'D' in the initial state, thread t3 terminates early at the initial state. So do t5, t7 and t8. On the other hand, thread t6 can match two patterns, one is "AB" and the other is "ABG". As PFAC delivers the longest pattern, "ABG" is accepted because "AB" is prefix of "ABG".

1.2 Performance of the PFAC algorithm

In the following, we briefly introduce the experimental results which show that using the PFAC library achieves significant performance enhancements compared to the traditional Aho-Corasick algorithm implemented on CPUs. The experimental setup has two machines: the *host* machine and the *device* machine. The host is equipped with an Intel® Core™ i7-950 running the Linux X86_64 operating system with 12GB DDR3 memory on an ASUS P6T-SE motherboard while the device is equipped with an Nvidia® GeForce® GTX480 GPU in the same Core™ i7 system with Nvidia driver version 260.19.29 and the CUDA 3.2 version. The test patterns are extracted from Snort V2.8 which contains 1,998 exact string patterns of total 41,997 characters. The length of the string pattern varies between one to 243 characters long. The total number of states is 27,754. The PFAC engine is tested using pure packets, and DEFCON packets. The pure packets which do not contain any patterns are used to evaluate the ideal throughput. The DEFCON packets which contain large amounts of real attack patterns are widely used to test commercial NIDS system. The size of test packets varies from 2 MB (Mega Bytes) to 192 MB. In order to compare the performance of the proposed algorithm, we re-implement three CPU versions and one GPU version as follows:

(1) AC_{CPU} : implementation of the AC algorithm on the Core™ i7 using single thread without any GCC optimization option.

(2) $DPAC_{OMP}$: implementation of the DPAC algorithm described in Section III on the Core™ i7 with OpenMP [2] library. The OpenMP library is a multi-threaded library used on multicore CPUs to achieve optimum performance. Moreover, because the Core™ i7 processor is a quad-core CPU with 2-way hyper-threading, the best performance comes from running 8 threads on 4 cores. Therefore, the $DPAC_{OMP}$ divides input streams into 8 segments and creates 8 threads to each segment for pattern matching by traversing the AC state machine.

(3) $PFAC_{OMP}$: implementation of the PFAC algorithm on the Core™ i7 with OpenMp library. Although $PFAC_{OMP}$ creates multiple threads whose number is equal to the input length, still maximum 8 threads can be activated on the Core™ i7 at one time. To achieve the best performance, dynamic scheduling is applied to schedule the multiple threads.

(4) $PFAC_{GPU}$: implementation of the PFAC algorithm on the GTX480. In $PFAC_{GPU}$, the best performance is achieved by creating 256 threads per block which allows 48 warps, 6 thread blocks per multiprocessor to be activated simultaneously.

For processing the DEFCON packets of 192MB, $PFAC_{GPU}$ achieves maximum raw data throughput of 122.84 Gbps while AC_{CPU}, $DPAC_{OMP}$, and $PFAC_{OMP}$ achieves 0.82, 3.28, and 3.67 Gbps. Compared to AC_{CPU}, $DPAC_{OMP}$, and $PFAC_{OMP}$, $PFAC_{GPU}$ achieves 150x, 37x, and 33x times speedup, respectively. As shown in Figure 4, $PFAC_{GPU}$ significantly outperforms to the AC_{CPU}, $DPAC_{OMP}$, and $PFAC_{OMP}$ approaches.

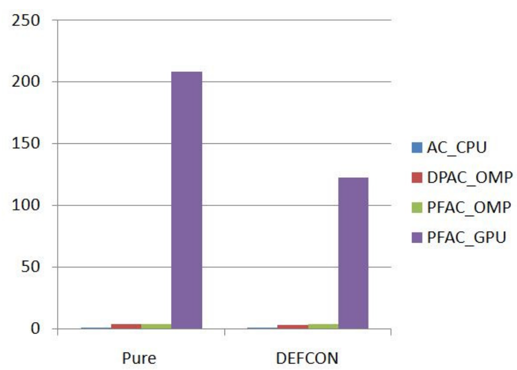

Figure 4: The raw data throughput of AC_{CPU}, $DPAC_{OMP}$, $PFAC_{OMP}$, and $PFAC_{GPU}$ over pure and DEFCON packets of 192MB

1.3 Introduction of PFAC library

PFAC library which is self-contained at API level utilizes the power of GPUs. An user needs not know anything of GPU programming, all he has to do is to create a PFAC handle (PFAC_create())

and to follow several APIs for loading patterns from file (PFAC_readPatternFromFile()), preparing input stream on host memory (CPU side), and processing input stream by calling PFAC_matchFromHost(). Finally PFAC would automatically transfer data to the device memory (GPU side), process and copy matched results back to the host memory. The basic model is shown as follows.

```
#include <PFAC.h>
...
PFAC_status = PFAC_create( &handle );
PFAC_status = PFAC_readPatternFromFile( handle, patternFile);
// prepare input stream in h_inputString
PFAC_status = PFAC_matchFromHost( handle, h_inputString, input_size, h_matched_result );
// process matched result in h_matched_result
```

For example, Figure 5 shows *h_matched_result* of the example in Figure 3. The number of *h_matched_result[j]* indicates the matched pattern at position *j* of input steam if *h_matched_result[j]* is nonzero.

position	0	1	2	3	4	5	6	7	8
h_matched_result	1	3	4	0	4	0	2	0	0

Figure 5: matched result of input stream "ABEDEDABG".

Sometimes users may only focus on matched results, so PFAC also provide a variant PFAC_matchFromHostReduce() which compresses *h_matched_result* and removes all zeros. The PFAC_matchFromHostReduce function provides match results in two arrays (*h_pos*, *h_matched_result*) where the *h_matched_result[j]* array stores the number of the matched pattern at the position *h_pos[j]* array.

```
#include <PFAC.h>
...
PFAC_status = PFAC_create( &handle );
PFAC_status = PFAC_readPatternFromFile( handle, patternFile);
// prepare input stream in h_inputString
PFAC_status = PFAC_matchFromHostReduce( handle, h_inputString, input_size,
    h_matched_result, h_pos, &h_num_matched ) ;
// process matched result in h_pos and h_matched_result
```

Figure 6 shows (*h_pos*, *h_matched_result*) of the same example in Figure 3. Because only five of the nine threads match, both size of *h_pos* and *h_matched_result* is 5, which is contained in the host variable *h_num_matched.*

h_pos	0	1	2	4	6
h_matched_result	1	3	4	4	2

Figure 6: compressed result of PFAC_matchFromHostReduce()

We would like to thank the Thrust project (http://code.google.com/p/thrust/) which is adopted in

our project to filter zeros out by inclusive-scan.
Please refer to Appendix A for the complete example.

PFAC types

The PFAC library has these types:

PFAC_handle_t

This is a pointer type to an opaque PFAC context, which has to be initialized by calling the PFAC_create() function prior to any other library functions. The definition of PFAC context is in $(PFAC_LIB_ROOT)/include/PFAC_P.h. However it is unnecessary to known the PFAC context if you are a regular user. In order to support different GPU architectures (sm11, sm12, sm13, sm20, sm21), PFAC is compiled into several dynamic modules in $(PFAC_LIB_ROOT)/lib, including libpfac_sm11.so, libpfac_sm12.so, libpfac_sm13.so, libpfac_sm20.so and libpfac_sm21.so. PFAC will load proper module according to the device type returned by the cudaGetDevice() function, a primitive function of CUDA library. PFAC library would not call cudaSetDevice(). If users want to bind some specific GPU, the cudaSetDevice() function has to be called explicitly. It is noted that the PFAC library only binds a handle to a GPU. If users want tp use multiple GPUs simultaneously, OpenMP library can be adopted to bind multiple handles to multiple GPUs. Please check the OpenMP example in $(PFAC_LIB_ROOT)/test/omp_PFAC.cpp. The handle created by PFAC_create() must be passed to every PFAC functions.

PFAC_platform_t

This type indicates the platform when running PFAC library. Because the PFAC algorithm works on many-core system, including CPU and GPU, a PFAC context keeps transition table on both CPU side and GPU side. An user can specify which side he wants to perform matching process by calling PFAC_setPlatform(). If PFAC_setPlatform() is not called, the PFAC library would use GPUs as the default platform. If PFAC_PLATFORM_CPU is passed as a parameter of PFAC_setPlatform(), the a single-thread version of PFAC would be activated on CPUs. If PFAC_PLATFORM_CPU_OMP is set, the PFAC library would run with multi-thread OpenMP version. The number of threads must be specified in the environment variable OMP_NUM_THREADS. Otherwise PFAC goes back to the single-thread version .

```
typedef enum {
    PFAC_PLATFORM_GPU = 0,    // default
```

```
    PFAC_PLATFORM_CPU = 1,
    PFAC_PLATFORM_CPU_OMP = 2
}PFAC_platform_t ;
```

PFAC_textureMode_t

This type indicates whether the transition table should be bound to 2-D texture memory or not. Suppose N denotes number of states of PFAC state machine and C denotes the number of ASCII characters (256). The transition table is allocated as a 2-D matrix of size N x 256. It is not space-efficient because many elements of the matrix is empty. We will provide a space-efficient version in the future release. An user can bind the transition table to 2-D texture memory explicitly by calling PFAC_setTextureMode(handle, PFAC_TEXTURE_ON). In our experiments, texture cache will bring 10% performance gain in average. However the 2-D texture memory has hardware limitation, 65536 x 32768 for sm1x or 65536 x 65535 for sm2x. Please check appendix G of CUDA programming guide. If N > 65536, the texture binding will fail and PFAC would return error. If PFAC_AUTOMATIC (default value) is set, PFAC would try to bind texture memory first, if binding fails, PFAC would bind table to linear memory.

```
typedef enum {
    PFAC_AUTOMATIC    = 0,    // default
    PFAC_TEXTURE_ON = 1,
    PFAC_TEXTURE_OFF = 2
}PFAC_textureMode_t ;
```

PFAC_perfMode_t

This type indicates whether timing-efficient algorithm or space-efficient algorithm is called at run-time. PFAC library has four main functions, matchFromDevice(), matchFromHost(), matchFromDeviceReduce() and matchFromHostReduce() and implements only one algorithm on matchFromDevice(), matchFromHost(). So type PFAC_perfMode_t only affects matchFromDeviceReduce() and matchFromHostReduce(). If PFAC_TIME_DRIVEN is set, then PFAC will allocate additional buffers to do computation, and space overhead is 2x more than space-efficient version. If users have not enough device memory to process a large input stream, then PFAC_SPACE_DRIVEN is a choice or users can split input stream into small ones and use multiple GPUs to process many small input stream at the same time.

```
typedef enum {
    PFAC_TIME_DRIVEN = 0, // default
    PFAC_SPACE_DRIVEN = 1
}PFAC_perfMode_t;
```

PFAC_status_t

This is a status type returned by library functions. PFAC_status_t has following values:

```
typedef enum {
    PFAC_STATUS_SUCCESS = 0 ,
    PFAC_STATUS_BASE = 10000,
    PFAC_STATUS_ALLOC_FAILED,
    PFAC_STATUS_CUDA_ALLOC_FAILED,
    PFAC_STATUS_INVALID_HANDLE,
    PFAC_STATUS_INVALID_PARAMETER,
    PFAC_STATUS_PATTERNS_NOT_READY,
    PFAC_STATUS_FILE_OPEN_ERROR,
    PFAC_STATUS_LIB_NOT_EXIST,
    PFAC_STATUS_ARCH_MISMATCH,
    PFAC_STATUS_INTERNAL_ERROR
} PFAC_status_t ;
```

The status values are explained below:

PFAC_STATUS_SUCCESS

The operation is done successfully.

PFAC_STATUS_BASE

It is used to separate CUDA error code and PFAC error code but align PFAC_STATUS_SUCCESS to cudaSuccess.

PFAC_STATUS_ALLOC_FAILED

Resource allocation fails on CPU side inside the PFAC library. For example, if the number of states of the PFAC state machine is huge, the C primitive function, malloc() may fail to allocate a space on CPU for storing the 2-D transition table.

PFAC_STATUS_CUDA_ALLOC_FAILED

Resource allocation fails on GPU side inside the PFAC library. if the number of states of the PFAC state machine is huge, the CUDA primitive function, cudaMalloc() may fail to allocate a space on GPU for storing the 2-D transition table.

PFAC_STATUS_INVALID_HANDLE

Users pass wrong handle to PFAC library functions. This is usually occurred when a NULL handle is passed.

PFAC_STATUS_INVALID_PARAMETER

Users pass wrong parameters to PFAC library functions. For example, wrong types of PFAC_platform_t or PFAC_textureMode_t are passed to PFAC library functions.

PFAC_STATUS_PATTERNS_NOT_READY

This happens when calling PFAC_matchFrom[Device,Host] prior to reading patterns by PFAC_readPatternFromFile().

PFAC_STATUS_FILE_OPEN_ERROR

Pattern file does not exist when calling PFAC_readPatternFromFile().

PFAC_STATUS_LIB_NOT_EXIST

Please check environment variable LD_LIBRARY_PATH. If the PFAC library is installed in /opt/PFAC (PFAC_LIB_ROOT=/opt/PFAC), the path of LD_LIBRARY_PATH has to be set as /opt/PFAC/lib. This is because PFAC is compiled into several dynamic modules in $(PFAC_LIB_ROOT)/lib, including libpfac_sm11.so, libpfac_sm12.so, libpfac_sm13.so, libpfac_sm20.so and libpfac_sm21.so. PFAC will load proper module according to the device type returned by cudaGetDevice(). If the path of the dynamic module is not set in LD_LIBRARY_PATH, the PFAC library cannot find a proper dynamic module.

PFAC_STATUS_ARCH_MISMATCH

PFAC library does not support sm10 because 32-bit atomic operation on global memory is used..

PFAC_STATUS_INTERNAL_ERROR

An internal PFAC operation failed. If programmers confirm size of all pre-allocated memory block, then please report bugs to authors.

Platform

PFAC does not support Microsoft windows system; it is tested on Fedora10 x86_64 (gcc version 4.3.2), Ubuntu 10.04 x86_64 (gcc version 4.4.3) and Mac OS X 10.6.6 i386 (gcc version 4.2.1 (Apple Inc. build 5664)). PFAC also requires CUDA toolkit, of which CUDA 3.2 version is recommended. You can download cuda toolkit 3.2 from http://developer.nvidia.com/object/cuda_3_2_downloads.html.

You must have at least one CUDA-enable Nvidia Graphic card with compute capability higher than 1.0 (sm1.0). PFAC library has tested on six GPUs listed in Table 1. These GPUs do not include sm1.2 and sm2.1, however programmers can still run PFAC library on sm1.2 and sm2.1. Please report bugs if any problem appears on sm1.2 and sm2.1.

Table 1: PFAC library has tested on five GPUs

	Compute capability	Number of multiprocessors	Number of CUDA cores	Total Global memory
C2070	2.0	14	448	6GB
GTX480	2.0	15	480	1.5GB
C1060	1.3	30	240	4GB
GTX260	1.3	24	192	895 MB
8800GT	1.1	14	112	511 MB
Geforce 9400M	1.1	2	16	256 MB

Installation

Step 1: download the PFAC library from http://code.google.com/p/pfac/

tar xzvf PFAC.tar.gz

In order to simplify notation, we define PFAC_LIB_ROOT as the top directory of the PFAC library (For example, if the PFAC library is installed in directory /opt/PFAC, PFAC_LIB_ROOT=/opt/PFAC). PFAC adopts recursive make, and the file layout is shown in Figure 7. Default compiler is g++ which is specified in common.mk. If you want to use Intel C++ compiler or some 3rd party compiler, please edit common.mk, modify variable CC, CXX and LINK. Second, the PFAC library also uses OpenMP to run CPU version, so it is necessary to link with OpenMP-support library.

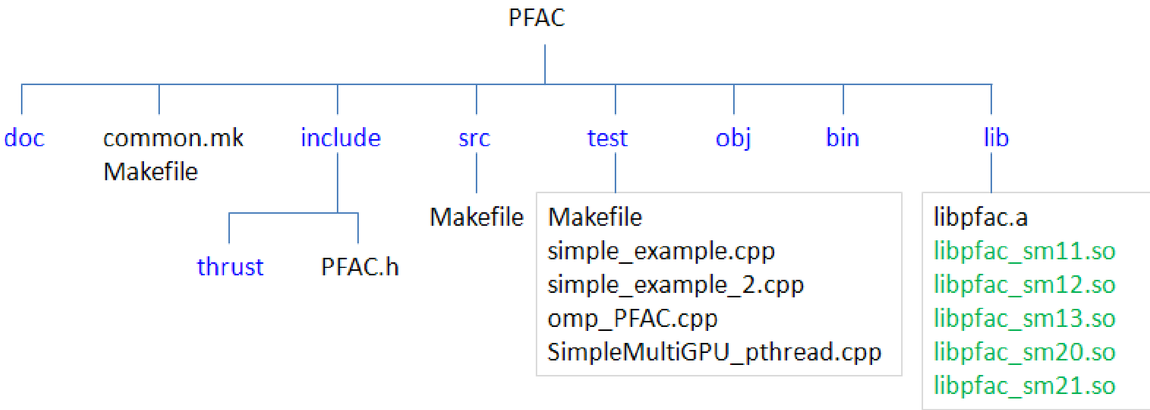

Figure 7: File layout of PFAC library

Note 1: PFAC library uses g++ -fPIC to generate dynamic module.

Note 2: PFAC library uses Thrust to do prefix-sum, the Thrust is included in $(PFAC_LIB_ROOT)/include, one can download Thrust from http://code.google.com/p/thrust/

Note 3: directory *doc* contains two files, one is PFAC_userGuide.pdf, the other is PFAC_algorithm.pdf which is a preprint paper [1].

Step 2: check if CUDA toolkit is installed

Users can download CUDA toolkit 3.2 from http://developer.nvidia.com/object/cuda_3_2_downloads.html.

The default installation directory of CUDA toolkit is /usr/local/cuda, to simplify discussion, CUDA_ROOT denotes top directory of CUDA toolkit.

If CUDA toolkit is ready, then try to find command nvcc by

which nvcc

If path of nvcc is not displayed, then check **PATH** environment variable, it should contains $(CUDA_ROOT)/bin. Users can modify $(HOME)/.bash_profile and modify **PATH**, **LD_LIBRARY_PATH/DYLD_LIBRARY_PATH** environment variables by following commands:

Linux:

```
export PATH=/usr/local/cuda/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda/lib:$LD_LIBRARY_PATH
```

Mac OS:

```
export PATH=/usr/local/cuda/bin:$PATH
export DYLD_LIBRARY_PATH=/usr/local/cuda/lib:$DYLD_LIBRARY_PATH
```

If machine is 64-bit, then change /usr/local/cuda/lib to /usr/local/cuda/lib64

Note 1: CUDA toolkit needs not to install in default directory, Makefile will automatically find $(CUDA_ROOT) by path of command nvcc.
Note 2: we test PFAC library on CUDA 3.0, everything is O.K. except that OpenMP example ($(PFAC_LIB_ROOT)/test/omp_PFAC.cpp) will hang on cudaMalloc() under some combination of multiple GPUs. So CUDA toolkit 3.2 is recommended.

Step 3: generate PFAC library by

    make

Then one static library (libpfac.a) and five dynamic modules (libpfac_sm11.so, libpfac_sm12.so, libpfac_sm13.so, libpfac_sm20.so, libpfac_sm21.so) will be generated in directory $(PFAC_LIB_ROOT)/lib.
Note: sm1.0 is not supported and PFAC library does not test on sm1.2 and sm2.1

Step 4: add $(PFAC_LIB_ROOT)/lib to **LD_LIBRARY_PATH** environment variable
Example: add following two lines in $(HOME)/.bash_profile
Linux:

    export LD_LIBRARY_PATH=$(PFAC_LIB_ROOT)/lib:$LD_LIBRARY_PATH

Mac OS:

    export DYLD_LIBRARY_PATH=$(PFAC_LIB_ROOT)/lib:$DYLD_LIBRARY_PATH

Note: PFAC library will load proper dynamic module at runtime. If $(PFAC_LIB_ROOT)/lib is not set in LD_LIBRARY_PATH, the PFAC library cannot find dynamic module, and report error code PFAC_STATUS_LIB_NOT_EXIST.

Step 5: verify PFAC library

    cd bin
    ./ simple_example.exe

```
At position    0, match pattern 1
At position    1, match pattern 3
At position    2, match pattern 4
At position    4, match pattern 4
At position    6, match pattern 2
```

If no error occurs, then PFAC library works.

Step 6: how to link PFAC library

$(PFAC_LIB_ROOT)/include/PFAC.h is a public header file, programmers must include this header file in the C/C++ source code. To compile the source codes, you must inform C/C++ compiler where PFAC.h locates by option -I$(PFAC_LIB_ROOT)/include. To link the object codes, you must provide static library $(PFAC_LIB_ROOT)/lib/libpfac.a by option -L$(PFAC_LIB_ROOT)/lib -lpfac. Moreover you need to link with OpenMP-support library (option -fopenmp is enough for g++ and option -openmp is for Intel C++ compiler).

For example:

g++ -fopenmp -I$(PFAC_LIB_ROOT)/include -L$(PFAC_LIB_ROOT)/lib -lpfac [your source code]

or

icpc -openmp -I$(PFAC_LIB_ROOT)/include -L$(PFAC_LIB_ROOT)/lib -lpfac [your source code]

What you must know

1. PFAC library provides C-style API. Programmers can link the library to C/C++ code or other language.

2. PFAC library uses 32-bit atomic operation which does not support compute capability 1.0 (sm1.0)

Table 2: features and technical specifications in appendix G of CUDA programming guide [3].

	Compute capability				
Feature support	**1.0**	**1.1**	**1.2**	**1.3**	**2.x**
Integer atomic functions operating on 32-bit words in global memory (Section B.11)	No	Yes			

3. PFAC library is optimized on Fermi (mainly sm2.0) and is functional correct on sm1.1, sm1.2, and sm1.3 but may not perform better on sm1.x.

4. PFAC library does not test on sm1.2 and sm2.1.

5. PFAC library accepts patterns of any length but performs better if length of pattern is smaller than 512. Moreover problem of line termination may occurs because Windows file systems use a two-character sequence carriage return followed by line feed (or CRLF) to terminate each line of a text file. If users take pattern file from Windows system, then it may contains carriage

return. Users can use UNIX utility functions /usr/bin/dos2unix and /usr/bin/unix2dos to correct carriage return.

6. PFAC context binds to only one GPU context and only default stream is used in PFAC library. Programmers can bind multiple PFAC contexts to multiple GPU by OpenMP library or pThread library, please see OpenMP example in $(PFAC_LIB_ROOT)/test/omp_PFAC.cpp or pThread example in $(PFAC_LIB_ROOT)/test/SimpleMultiGPU_pthread.cpp

7. programmers must always check error code returned by API because PFAC library does not record last error. If returned error code is not PFAC_STATUS_SUCCESS, then decode error code by helper function PFAC_getErrorString().
Basic procedure is as following

```
#include <PFAC.h>
....

PFAC_handle_t handle ;
PFAC_status_t PFAC_status ;

PFAC_status = PFAC_create( &handle ) ;

if ( PFAC_STATUS_SUCCESS != PFAC_status ){
    printf("Error: %s\n", PFAC_getErrorString(PFAC_status) );
    exit(1) ;
}
```

2. PFAC functions

This chapter discusses the PFAC functions, which are divided into two categories.
Category 1: PFAC Helper functions
Category 2: PFAC main functions

PFAC Helper Functions

The purpose of PFAC helper functions is to provide a tool to control and debug the library, They are as follows:

Function PFAC_create()

PFAC_status_t

PFAC_create(PFAC_handle_t ****handle*)**

Initializes the PFAC library and creates a handle on the PFAC context. The handle must be created before any other PFAC API calls. This function would load dynamic module according to GPU types from cudaGetDevice(). The PFAC library does not support compute capability 1.0.

Note: if users have multiple GPU and want to bind a specific GPU to PFAC library, CUDA device must be set first by calling cudaSetDevice(), a primitive CUDA function.

Output

Handle	A pointer pointing to a PFAC context

Status returned

PFAC_STATUS_SUCCESS	PFAC library initializes successfully
PFAC_STATUS_ALLOC_FAILED	PFAC context cannot be created
PFAC_STATUS_ARCH_MISMATCH	PFAC library uses 32-bit atomic operation which does not support compute capability 1.0
PFAC_STATUS_LIB_NOT_EXIST	Please check LD_LIBRARY_PATH environment variable. It must contain $(PFAC_LIB_ROOT)/lib
PFAC_STATUS_INTERNAL_ERROR	Please report bugs

Function PFAC_destroy()

PFAC_status_t

PFAC_destroy(PFAC_handle_t *handle*)

Releases CPU side resources and GPU side resources used by the PFAC library.

Input

Handle	Handle to a PFAC context

Status returned

PFAC_STATUS_SUCCESS	PFAC context is de-allocated successfully
PFAC_STATUS_INVALID_HANDLE	if *handle* is a NULL pointer

Function PFAC_setPlatform()

PFAC_status_t

PFAC_setPlatform(PFAC_handle_t *handle*, PFAC_platform_t *platform*)

The main goal of the PFAC library is to accelerate string matching on the GPU side. However it can support the same functionality on the CPU side because PFAC is a generic many-core algorithm. One can configure the PFAC library on the CPU side by PFAC_PLATFORM_CPU (single thread) or PFAC_PLATFORM_CPU_OMP (multiple threads).

Note 1: PFAC uses OpenMP to do parallel computation. User must set OMP_NUM_THREADS environment variable to invoke OpenMP version.

Note 2: even users configure PFAC to perform string matching on CPUs, PFAC library would still allocate resources on GPUs when users call PFAC_readPatternFromFile(). In other words, the PFAC library prepares two copies of transition table on both the CPU side and the GPU side.

Input

handle	Handle to a PFAC context
platform	PFAC_PLATFORM_GPU (default) , PFAC_PLATFORM_CPU or PFAC_PLATFORM_CPU_OMP

Status returned

PFAC_STATUS_SUCCESS	Operation is successful
PFAC_STATUS_INVALID_HANDLE	if *handle* is a NULL pointer
PFAC_STATUS_INVALID_PARAMETER	if *platform* is not PFAC_PLATFORM_GPU , PFAC_PLATFORM_CPU or PFAC_PLATFORM_CPU_OMP

Function PFAC_setTextureMode()

PFAC_status_t

PFAC_setTextureMode(PFAC_handle_t *handle*, PFAC_textureMode_t *textureModeSel*)

Default value of texture mode is PFAC_AUTOMATIC. Under this setting, PFAC library would try to bind transition table to a 2-D texture memory. If binding fails (for example, transition table is too large to fit into 2-D texture), the PFAC library binds table to linear memory. If users disable texture binding by the parameter PFAC_TEXTURE_OFF, the PFAC library would bind transition table to the CUDA linear memory. Users can also enforce PFAC library to bind texture by the parameter PFAC_TEXTURE_ON. However the PFAC library will return error if binding fails and will not bind to linear memory.

Input

Handle	Handle to a PFAC context
textureModeSel	PFAC_AUTOMATIC (default), PFAC_TEXTURE_ON or PFAC_TEXTURE_OFF

Status returned

PFAC_STATUS_SUCCESS	Operation is successful
PFAC_STATUS_INVALID_HANDLE	if *handle* is a NULL pointer
PFAC_STATUS_INVALID_PARAMETER	if *textureModeSel* is not PFAC_AUTOMATIC, PFAC_TEXTURE_ON or PFAC_TEXTURE_OFF
PFAC_STATUS_CUDA_ALLOC_FAILED	Either texture binding fails or allocation of linear memory fails
PFAC_STATUS_INTERNAL_ERROR	Please report bugs

Function PFAC_setPerfMode()

PFAC_status_t

PFAC_setPerfMode(PFAC_handle_t handle, PFAC_perfMode_t perfModeSel)

PFAC library provides time-efficient and space-efficient versions for functions matchFromHostReduce() and matchFromDeviceReduce(). Programmers can choose desired version by performance mode, either PFAC_TIME_DRIVEN or PFAC_SPACE_DRIVEN. Default value of performance mode is PFAC_TIME_DRIVEN. Memory requirement of time-efficient version is 2x than space-efficient version. Table 4 and Table 5 shows estimation of memory requirement.

Note 1: sometimes space-efficient version performs better than time-efficient version.

Note 2: PFAC_matchFromDevice() and PFAC_matchFromHost() have only time-efficient version, space-efficient version will be ready in the future release.

Input

handle	Handle to a PFAC context
perfModeSel	PFAC_TIME_DRIVEN (default) or PFAC_SPACE_DRIVEN

Status returned

PFAC_STATUS_SUCCESS	Operation is successful

PFAC_STATUS_INVALID_HANDLE	if *handle* is a NULL pointer
PFAC_STATUS_INVALID_PARAMETER	if *perfModeSel* is not PFAC_TIME_DRIVEN, or PFAC_TIME_DRIVEN

Function PFAC_getErrorString()

const char*

PFAC_getErrorString(PFAC_status_t *status*)

Returns the message string from error code *status* which is returned error code of API.

Input

status	error code from API

Returns

char* pointer to a NULL-terminated string. This is a string literal, do not overwrite it.

Function PFAC_dumpTransitionTable()

PFAC_status_t

PFAC_dumpTransitionTable(PFAC_handle_t *handle*, FILE **fp*)

Outputs transition table to a file pointed by *fp*.

The Format of the transition table includes two parts.

Part 1: valid transition represented by

(current state, input character) -> next state

If a numerical value of input character is between 0x20 and 0x7E, the input character is human readable, and is printed using %c, otherwise is printed using %x.

Part 2: relationship between final states and patterns

[final state] [matched pattern ID] [pattern length] [pattern(string literal)]

Example: The transition table of the PFAC state machine in Figure 1 is shown as follows. One can obtain this table by executing simple_example.cpp in Appendix A.

```
# Transition table: number of states = 11, initial state = 5
# (current state, input character) -> next state
(    1,    G) -> 2
(    5,    A) -> 6
(    5,    B) -> 7
(    5,    E) -> 10
(    6,    B) -> 1
(    7,    E) -> 8
(    8,    D) -> 9
(    9,    E) -> 3
```

```
(    10,      D) -> 4
# Output table: number of final states = 4
# [final state] [matched pattern ID] [pattern length] [pattern(string literal)]
      1         1         2        "AB"
      2         2         3        "ABG"
      3         3         4        "BEDE"
      4         4         2        "ED"
```

Input

handle	Handle to a PFAC context
fp	File handler which is opened by caller. If *fp* is a NULL pointer, then redirect it to standard output.

Status returned

PFAC_STATUS_SUCCESS	Operation is successful
PFAC_STATUS_INTERNAL_ERROR	Please report bugs

PFAC main functions

The PFAC main functions are as follows:

PFAC_readPatternFromFile( ) on page 22
PFAC_matchFromDevice() on page 23
PFAC_matchFromHost() on page 24
PFAC_matchFromDeviceReduce() on page 25
PFAC_matchFromHostReduce() on page 27

All CUDA kernels (launched by triple bracket <<<, >>>) and some CUDA API functions are asynchronous: Control is returned to the host thread before the device has completed the required task (section 3.2.7 in CUDA programming guide [3]). PFAC library has 4 main functions listed in Table 3, only one among them is asynchronous because three synchronous functions use CUDA API cudaMemcpy() , cudaMalloc() or cudaFree() which are synchronous.

Table 3: Asynchronous/Synchronous of four main functions

	Synchronous	**Asynchronous**
PFAC_matchFromDevice() and PFAC_TEXTURE_OFF		X
PFAC_matchFromDevice() and PFAC_TEXTURE_ON	X	
PFAC_matchFromHost()	X	
PFAC_matchFromDeviceReduce()	X	
PFAC_matchFromHostReduce()	X	

Programmers must always check error code of type PFAC_status_t which is returned by API call. However error code returned by asynchronous function PFAC_matchFromDevice() is tricky when texture mode is PFAC_TEXTURE_OFF. Even the error code is PFAC_STATUS_SUCCESS, the function may not be correct. Because PFAC_matchFromDevice() is asynchronous when PFAC_TEXTURE_OFF is set, then it will return error code before completion of built-in CUDA kernel. The error code PFAC_STATUS_SUCCESS means that no invalid function arguments or internal states are found, not success of built-in CUDA kernel. For example, the input parameters of PFAC_matchFromDevice(), *d_inputString* and *d_matched_result* reside in device memory and are allocated by users. If input stream has N bytes, then users must allocate d_inputString of N bytes and d_matched_result of 4*N bytes, also pass N to parameter *size*. However PFAC_matchFromDevice() cannot check consistency between parameters *size* and (*d_inputString*, *d_matched_result*). For example, in Figure 8, PFAC_matchFromDevice() is called at time T1 and returns error code at time T2. At the same time, kernel is launched on GPU side. However out-of-array-bound occurs at time T3 (during kernel execution), and host code (CPU side) does not know this error. If programmers want to check validity of PFAC_matchFromDevice(), then following code is a choice

```
....
assert( PFAC_STATUS_SUCCESS == PFAC_setTextureMode( handle, PFAC_TEXTURE_OFF) ) ;
assert( PFAC_STATUS_SUCCESS == PFAC_matchFromDevice( handle,
        d_inputString, size, d_matched_result) );
cudaThreadSynchronize();
assert( cudaSuccess == cudaGetLastErro());
```

The flow chart of above code is shown in Figure 9.

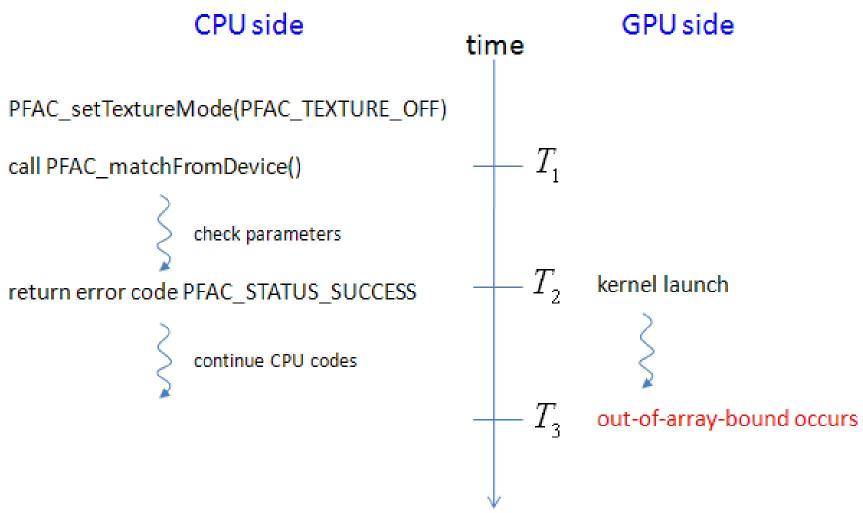

Figure 8: asynchronous nature of PFAC_matchFromDevice() with non-texture mode.

The other alternative is to set **CUDA_LAUNCH_BLOCKING** environment variable suggested in section 3.2.7.1 of CUDA programming guide [3].

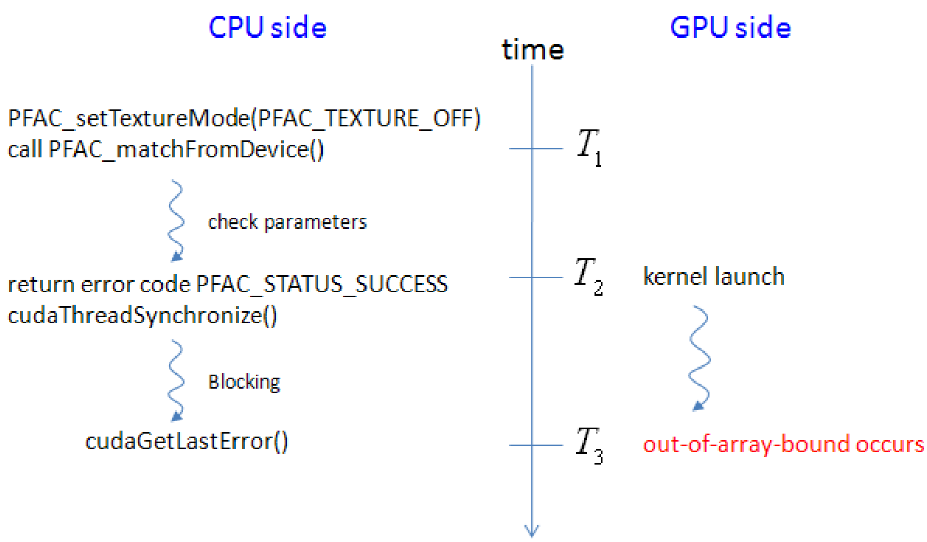

Figure 9: use cudaThreadSynchronous() to synchronize PFAC_matchFromDevice().

Note 1: PFAC_matchFromDevice() with texture mode (PFAC_TEXTURE_ON) will bind/unbind texture memory by API cudaBindTextureToArray()/cudaUnbindTexture(), so it is synchronous.

Note 2: default value of texture mode is PFAC_AUTOMATIC, if PFAC library cannot bind texture memory, then PFAC_matchFromDevice() will be asynchronous.

Function readPatternFromFile

PFAC_status_t

PFAC_readPatternFromFile(PFAC_handle_t *handle*, char **filename*)

Parses pattern file *filename*, and creates transition table both on the CPU side and the GPU side.

Note 1: the parser only accept ASCII characters (0~255) and cannot accept escape character. For example, \n is interpreted as two characters, one is back slash '\' and the other is 'n'. The parser identifies a pattern which is between two new lines. So if a pattern has a new line character (0x0A), then it becomes two patterns. For example, the pattern "abc\nd" whose numerical value is 0x61 0x62 0x63 0x0A 0x64 will becomes two patterns, one is "abc" and the other is "d".

Note 2: problem of line termination may occurs because Windows file systems use a two-character sequence carriage return followed by line feed (or CRLF) to terminate each line of a text file. If users take pattern file from Windows system, then it may contains carriage return. Users can use UNIX utility functions /usr/bin/dos2unix and /usr/bin/unix2dos to correct carriage return.

Note 3: The PFAC library stores the PFAC state machine as a 2-D table. This is memory inefficient. Suppose S denotes number of states, C denotes the number of ASCII characters (256), then S*C is size of transition table on GPU side. If N denotes the number of characters of file *filename*, the value of N*C is the size of the transition table on the CPU side. Memory efficient version would be released soon.

Note 4: The PFAC library can only accept one pattern file, if 2nd PFAC_readPatternFromFile() is called, then library would free transition table of previous pattern file and reconstruct transition table of second pattern file.

Input

handle	Handle to a PFAC context
filename	Name of a pattern file

Status returned

PFAC_STATUS_SUCCESS	Operation is successful
PFAC_STATUS_INVALID_HANDLE	The *handle* is a NULL pointer. Please call PFAC_create() to create a legal handle.
PFAC_STATUS_INVALID_PARAMETER	The *filename* is a NULL pointer. The Library does not support patterns from standard Input.
PFAC_STATUS_FILE_OPEN_ERROR	The file *filename* does not exist.
PFAC_STATUS_ALLOC_FAILED	The host (device)memory is not enough to parse

PFAC_STATUS_CUDA_ALLOC_FAILED	pattern file. The pattern file is too large to allocate host (device)memory. Please split the pattern file into smaller ones and try again.
PFAC_STATUS_INTERNAL_ERROR	Please report bugs.

Function PFAC_matchFromDevice()

PFAC_status_t

PFAC_matchFromDevice(PFAC_handle_t *handle*, char **d_inputString*, size_t *size*, int **d_matched_result*)

Given input stream *d_inputString* of *size* bytes in device memory, find pattern ID of each substring and store pattern ID into device memory *d_matched_result.*

d_matched_result[k] is id (id is nonzero) if prefix of *d_inputString[k,:]* matches the pattern of ID id.

Example: suppose pattern is {AB, ABG, BEDE, ED} and input stream is { ABEDEDABG}, then *d_matched_result* = {1,3,4,0,4,0,2,0,0}

please check Figure 1 and Figure 5.

Note 1: In order to decrease the penalty of memory accesses, the PFAC library reads integer instead of character when reading an input stream. If the size of the input stream is not multiples of 4, segmentation fault occurs logically but may not happen physically because basic unit of cudaMalloc is 256 bytes. It is recommended to allocate a device memory of size N which is equal to size - (size & 3) + 4 if the size of input stream is not multiples of 4.

Note 2: type of *size* is unsigned integer (size_t), if programmers pass a negative number, then *size* will becomes a HUGE positive number.

Input

handle	Handle to a PFAC context
d_inputString	Device memory of *size* bytes at least Input stream
size	Number of characters of input stream *d_inputString*

Output

d_matched_result	Device memory of 4 * *size* bytes at least If *d_matched_result[k]* is non-zero, then it is pattern ID starting at position *k* of *d_inputString*

Status returned

PFAC_STATUS_SUCCESS	Operation is successful.
PFAC_STATUS_INVALID_HANDLE	if *handle* is a NULL pointer. Please call PFAC_create() to create a legal handle.
PFAC_STATUS_INVALID_PARAMETER	if *d_inputString or d_matched_result* is a NULL pointer.

PFAC_STATUS_PATTERNS_NOT_READY	if patterns are not loaded first. Please call PFAC_readPatternFromFile() first.
PFAC_STATUS_INTERNAL_ERROR	check if *d_inputString* (*d_matched_result*) is allocated by *size* (4 **size*) bytes. if it is, please report bugs.

Function PFAC_matchFromHost()

PFAC_status_t

PFAC_matchFromHost(PFAC_handle_t *handle*, char **h_inputString*, size_t *size*, int **h_matched_result*)

The function is similar to the PFAC_matchFromDevice() function except that *h_inputString* and *h_matched_result* are stored in the host memory. The PFAC library allocates d_inputString and d_matched_result first, then copy h_inputString to d_inputString, call PFAC_matchFromDevice(), and finally copy d_matched_result back to h_matched_result.

Note: it is possible that the device memory is not enough to allocate both d_inputString and d_matched_result. The simple way is to divide input steam into small ones and call PFAC_matchFromHost() several times. It is noted that two adjacent small input streams must overlap at least M characters where M is maximum length of patterns.

Input

handle	Handle to a PFAC context
h_inputString	Host memory of *size* bytes at least Input stream
size	Number of characters of input stream *h_inputString*

Output

h_matched_result	Host memory of 4 * *size* bytes at least If *h_matched_result[k]* is non-zero, then it is pattern ID starting at position *k* of *h_inputString*

Status returned

PFAC_STATUS_SUCCESS	Operation is successful.
PFAC_STATUS_INVALID_HANDLE	The *handle* is a NULL pointer. Please call PFAC_create() to create a legal handle.
PFAC_STATUS_INVALID_PARAMETER	The *h_inputString or h_matched_result* is a NULL pointer.
PFAC_STATUS_PATTERNS_NOT_READY	The patterns are not loaded first. Please call PFAC_readPatternFromFile() first.
PFAC_STATUS_CUDA_ALLOC_FAILED	The device memory is not enough to allocate *d_inputString* and *d_matched_result*. Users must check memory usage of GPUs. If the memory is not

	enough, users must divide the input stream into small ones and call PFAC_matchFromHost() multiple times.
PFAC_STATUS_INTERNAL_ERROR	check if *h_inputString* (*h_matched_result*) is allocated by *size* (4 **size*) bytes. if it is, please report bugs.

Function PFAC_matchFromDeviceReduce()

PFAC_status_t

PFAC_matchFromDeviceReduce(PFAC_handle_t *handle*, char **d_inputString*, size_t *size*, int **d_matched_result*, int **d_pos*, int **h_num_matched*)

The function is similar to the PFAC_matchFromDevice() function except that the *d_matched_result* is compressed without zeros.

The data stored in *d_matched_result[k]* is id if prefix of *d_inputString[d_pos[k],:]* matches the pattern of ID id.

Example: suppose patterns are {AB, ABG, BEDE, ED} and the input stream is {ABEDEDABG}, then
h_num_matched = 5,
d_pos = {0, 1, 2, 4, 6} and
d_matched_result = {1, 3, 4, 4, 2}

Please check Figure 1 and Figure 6.

Note 1: *h_num_matched* is a scalar in host memory.

Note 2: PFAC library provides two versions on PFAC_matchFromDeviceReduce(), one is time-efficient and the other is space-efficient. Programmers can choose time-efficient version by PFAC_setPerfMode(PFAC_TIME_DRIVEN) and space-efficient version by PFAC_setPerfMode(PFAC_SPACE_DRIVEN). The default value is PFAC_TIME_DRIVEN.

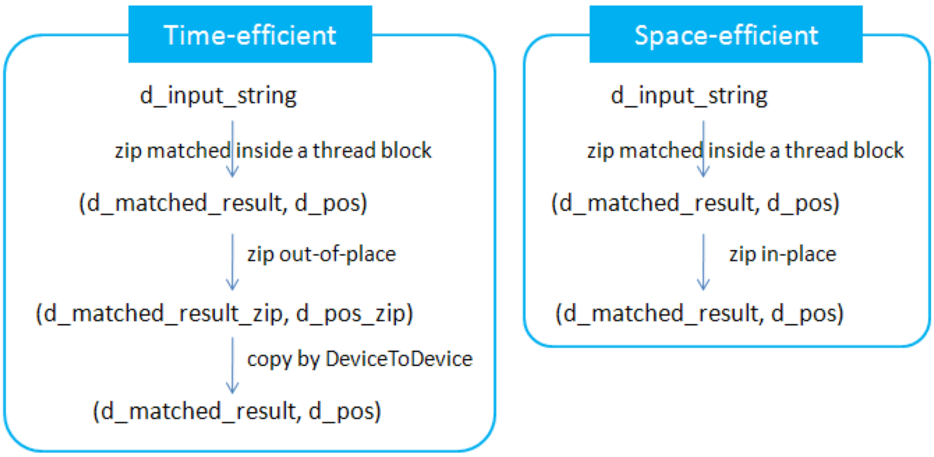

Figure 10: flow chart of PFAC_matchFromDevice()

The time-efficient version uses additional buffer to do compression, the flow chart is shown in

Figure 10. The difference between time-efficient and space-efficient is global compression, time-efficient uses buffer (d_matched_result_zip, d_pos_zip) to compress, i.e. out-of-place compression whereas space-efficient does in-place compression. Space-efficient version has race condition due to nature of in-place. Race condition is removed by global synchronization implemented by atomic operations.
Suppose N denotes number of bytes of input stream, the required space of time-efficient version is bounded by (17+1/128)*N bytes and space-efficient version is bounded by (9+3/256)*N bytes. Table 5 shows that if PFAC library binds to GTX480 which has 1.5GB device memory, then maximum size of input stream is 88 MB on time-efficient version and 166 MB on space-efficient version.

Table 4: total memory required by time-efficient (space-efficient) version when size of input stream varies.

N = size of input stream	8 MB	16 MB	32 MB	64 MB	128 MB	192 MB
Time-efficient	136 MB	272 MB	545 MB	1090 MB	2177 MB	3266 MB
Space-efficient	72 MB	144 MB	289 MB	577 MB	1153 MB	1730 MB

Table 5: maximum size of input stream on different GPUs.

	C2070 6GB ECC off	C2050 3GB ECC off	C1060 4 GB	GTX480 1.5 GB	GTX460 1 GB	GTX460 768 MB
Time-efficient	352 MB	176 MB	235 MB	88 MB	58 MB	45 MB
Space-efficient	666 MB	333 MB	444 MB	166 MB	111 MB	85 MB

Note 3: in our benchmarks, time-efficient version is not always faster than space-efficient version because time-efficient version requires extra data transfer (device to device). However time-efficient is still the default value in current release.
Note 4: PFAC uses Thrust (http://code.google.com/p/thrust/) to do prefix-sum. This operation requires extra device memory, roughly *size*/128 bytes. The extra memory is automatically allocated and de-allocated inside API.
Note 5: although *d_matched_result* is compressed without zeros and *d_pos* stores corresponding positions, they are also working space, and size of *d_matched_result* and *d_pos* must be 4*size bytes. If not, then returned error code may be PFAC_STATUS_INTERNAL_ERROR.

Input

handle	Handle to a PFAC context
d_inputString	Device memory of *size* bytes at least

	Input stream
size	Number of characters of input stream d_inputString

Output

d_matched_result	Device memory of 4**size* bytes at least *d_matched_result[k]* is matched pattern ID starting at position *d_pos[k]* of input stream *d_inputString*
d_pos	Device memory of 4**size* bytes at least Position of first character of a matched pattern in input stream
h_num_matched	Host memory, scalar Number of matched patterns

Status returned

The same as PFAC_matchFromDevice()

Function PFAC_matchFromHostReduce()

PFAC_status_t

PFAC_matchFromHostReduce(PFAC_handle_t *handle*, char **h_inputString*, size_t *size*, int **h_matched_result*, int **h_pos*, int **h_num_matched*)

The function is similar to the PFAC_matchFromDeviceReduce() function except that *h_inputString*, *h_matched_result* and *h_pos* are stored in the host memory.

Input

handle	Handle to a PFAC context
h_inputString	Host memory of *size* bytes at least input stream
size	Number of characters of input stream h_inputString

Output

h_matched_result	Host memory of 4**size* bytes at least *h_matched_result[k]* is matched pattern ID starting at position *h_pos[k]* of input string *h_inputString*
h_pos	Host memory of 4**size* bytes at least Position of first character of a matched pattern in input stream
h_num_matched	Host memory, scalar Number of matched patterns

Status returned

The same as PFAC_matchFromHost()

Reference

[1] Cheng-Hung Lin, Chen-Hsiung Liu, Lung-Sheng Chien, and Shih-Chieh Chang, "Accelerating Pattern Matching Using a Novel Parallel Algorithm on GPU," preprint. You can find it in directory $(PFAC_LIB_ROOT)/doc.

[2] OpenMP, Available: http://openmp.org/wp/

[3] NVIDIA CUDA C Programming Guide, Available: http://developer.nvidia.com/object/cuda_3_2_downloads.html

[4] Thrust library, available: http://code.google.com/p/thrust/

Appendix A PFAC example

Example 1 describes how to use PFAC library. The complete code is also listed in $(PFAC_LIB_ROOT)/test/simple_example.cpp. To simplify the code, we hard-code path of a pattern file and an input file, where pattern file can be found in $(PFAC_LIB_ROOT)/test/pattern/example_pattern and input file is in $(PFAC_LIB_ROOT)/test/data/example_input.

The example shows following operations:

1. Including the header file PFAC.h which resides in directory $(PFAC_LIB_ROOT)/include. This header file is necessary because it contains declaration of APIs.
2. Initializing the PFAC library by creating a PFAC handle (PFAC binds to a GPU context implicitly. If an user wants to bind a specific GPU, he must call cudaSetDevice() explicitly before calling PFAC_create()).
3. Reading patterns from a file and PFAC would create transition table both on the CPU side and the GPU side.
4. Dumping transition table to "table.txt", the content of table is shown in Figure 1.
5. Reading an input stream from a file.
6. Performing matching process by calling the PFAC_matchFromHost() function.
7. Showing matched results.
8. Destroying the PFAC handle.

PFAC library does not record last error, so users must check return error code themselves and use PFAC_getErrorString() to report error message.

There are five examples listed in directory $(PFAC_LIB_ROOT)/test, they are

filename	Purpose
simple_example.cpp	How to use PFAC_matchFromHost()
simple_example_reduce.cpp	How to use space-efficient PFAC_matchFromHostReduce()
profiling.cpp	Report timing, 4 combinations chosen from command line options 1) PFAC_matchFromDevice() + texture ON 2) PFAC_matchFromDevice() + texture OFF 3) PFAC_matchFromHost() + texture ON 4) PFAC_matchFromHost() + texture OFF
omp_PFAC.cpp	Given a pattern file and a input stream, uses all GPUs to process

	this input stream by OpenMP. Each thread processes a segment of input stream. A simple job scheduling is adopted in this demonstration. This example shows 1) how to use OpenMP to create multiple PFAC contexts and bind each context to different GPUs. 2) how to extract a segment of input stream and update corresponding result. 3) how to use static scheduling in a parallel section 4) how to solve "boundary detection" in [1] **WARNING 1**: LD_LIBRARY_PATH must contain dynamic module of OpenMP library. For example, /usr/lib64 in Fedora x86_64 **WARNING 2**: if users install CUDA 3.0, then OpenMP example will hang on cudaMalloc() under some combination of multiple GPUs. So CUDA toolkit 3.2 is recommended
SimpleMultiGPU_pthread.cpp	Two threads bind different PFAC contexts to two different GPUs. thread 0 processes (pattern 0, input 0) thread 1 processes (pattern 1, input 1) This example shows 1) how to use pthread_create() to create a thread and bind a thread to a task (a function pointer) 2) use pthread_join() to merge all threads

Example 1: using PFAC library

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <PFAC.h>

int main(int argc, char **argv)
{
    char dumpTableFile[] = "table.txt" ;
    char inputFile[] = "../test/data/example_input" ;
    char patternFile[] = "../test/pattern/example_pattern" ;
    PFAC_handle_t handle ;
    PFAC_status_t PFAC_status ;
    int input_size ;
    char *h_inputString = NULL ;
    int    *h_matched_result = NULL ;

    // step 1: create PFAC handle
    PFAC_status = PFAC_create( &handle ) ;
    assert( PFAC_STATUS_SUCCESS == PFAC_status );

    // step 2: read patterns and dump transition table
    PFAC_status = PFAC_readPatternFromFile( handle, patternFile) ;
    if ( PFAC_STATUS_SUCCESS != PFAC_status ){
        printf("Error: fails to read pattern from file, %s\n", PFAC_getErrorString(PFAC_status) );
        exit(1) ;
    }
    // dump transition table
    FILE *table_fp = fopen( dumpTableFile, "w") ;
    assert( NULL != table_fp ) ;
    PFAC_status = PFAC_dumpTransitionTable( handle, table_fp );
    fclose( table_fp ) ;
    if ( PFAC_STATUS_SUCCESS != PFAC_status ){
        printf("Error: fails to dump transition table, %s\n", PFAC_getErrorString(PFAC_status) );
        exit(1) ;
    }
```

Example 1: using PFAC library (continued)

```
//step 3: prepare input stream
FILE* fpin = fopen( inputFile, "rb");
assert ( NULL != fpin ) ;

// obtain file size
fseek (fpin , 0 , SEEK_END);
input_size = ftell (fpin);
rewind (fpin);

// allocate memory to contain the whole file
h_inputString = (char *) malloc (sizeof(char)*input_size);
assert( NULL != h_inputString );

h_matched_result = (int *) malloc (sizeof(int)*input_size);
assert( NULL != h_matched_result );
memset( h_matched_result, 0, sizeof(int)*input_size ) ;

// copy the file into the buffer
input_size = fread (h_inputString, 1, input_size, fpin);
fclose(fpin);

// step 4: run PFAC on GPU
PFAC_status = PFAC_matchFromHost( handle, h_inputString, input_size, h_matched_result ) ;
if ( PFAC_STATUS_SUCCESS != PFAC_status ){
    printf("Error: fails to PFAC_matchFromHost, %s\n", PFAC_getErrorString(PFAC_status) );
    exit(1) ;
}

// step 5: output matched result
for (int i = 0; i < input_size; i++) {
    if (h_matched_result[i] != 0) {
        printf("At position %4d, match pattern %d\n", i, h_matched_result[i]);
    }
}

PFAC_status = PFAC_destroy( handle ) ;
```

Example 1: using PFAC library (continued)

```
    assert( PFAC_STATUS_SUCCESS == PFAC_status );

    free(h_inputString);
    free(h_matched_result);

    return 0;
}
```
