KAGRA Algorithmic Library (KAGALI)

Present Status of Development and Application

Ken-ichi Oohara
Niigata University

on behalf of the KAGRA Data Analysis Subgroup
Introduction

- Software for gravitational wave data analysis: LSC has already opened a fine software suite, LALSuite, to the public.

- Nevertheless, we should prepare software proper to KAGRA, especially for data handling tools and for new methods we are developing.
Introduction

- LALSuite:
  - prepares a clever error handling/bug tracking system.
  - seems to be very complicated for us to append new functions.

- We decided to develop our own software suite.
Introduction

**KAGALI: KAGRA Algorithmic Library**

- **KAGALI**
  - a data analysis library that KAGRA data analysis subsystem is developing

- **KAGALI-Apps:**
  - a data analysis application software packages build upon KAGALI, LALSuite and libraries developed by Virgo.

- For KAGRA data analysis, we will use any of available software including KAGALI, LALSuite, Virgo software, and so on.
**KAGRA and KAGALI**

- **KAGURA 神楽** in Japanese means a kind of religious music and dance based on Shinto.

- **KAGARI 籐 or KAGARI-BI 篱火** means a bonfire or a watch fire.

- People usually build KAGARI-BI to light-up KAGUARA.
KAGURA 神楽 in Japanese means a kind of religious music and dance based on Shinto.

KAGARI 籐 or KAGARI-BI 籐火 means a bonfire or a watch fire.

We build KAGALI to light-up KAGRA.
KAGALI and KAGALI-Apps do not specify which language we have to use to develop them. But C-language is used in the major part now.

KAGALI is assumed to be installed and work on various systems of, at least, Unix-like OS including Linux and Mac OS X.

- We don’t place first priority on portability. We mainly intend to achieve higher performance.
- At present, it is confirmed for RHEL 6.x (7.x) and its compatibles (SL, CentOS) as well as Mac OS X, with GNU C compiler (gcc) and Intel C compiler (icc).
We set coding style guidelines
(KAGALI C-coding style guide, 2014/7/19)

- to reduce bugs
- to make code-debugging easy
- to produce a common properties by which the KAGRA data analysis team can produce data analysis programs efficiently and quickly
- to make programs easily re-usable and understandable by standardizing code appearances
Coding guidelines

• You should write a source code that is easy to understand for others and you-at-some-time-later.

• You should write a source code that is easy to understand without a comment.
Coding Guidelines

- KAGALI must be written in
  - clean C
  - C99

You may use GNU extensions if the performance is improved, but you must add the alternative part that works on standard C99; one of them will be chosen by the preprocessor.

```c
#ifdef __GNUC__
    (the GNU extensions (-std=gnu99) part)
#else
    (the C99 (-std=c99) part)
#endif
```
Coding Guidelines

- Library should not
  - call file input/output routines except for the libraries that handles Frame Lib.
  - call any system-dependent functions such as `system()`, `getenv()`, `srand()` etc.

- Functions in the library should not terminate the program, that is, should not call `abort()`, `exit()` or issue `signal()` except for some particular cases.
  - Functions should return a status code via the designated structure `KGLStatus`, instead.
Atomic Data Types

- We decided not to use the LAL data types specifiers (INT4, INT8, REAL4, REAL8 ...)

- Integer data types:
  Use `intN_t` (int32_t, int64_t etc) of C99 if clarification of data size is essential.

- Floating point data types:
  Use `float` and `double`.

- Boolean data type:
  Use `_Bool` in C99.
  You can use `bool`, `true`, `false` defined in the C99 header `stdbool.h`

- Complex data type:
  Ether of `_Complex` defined in C99 or `complex` defined in `complex.h` can be used.
Name Space

- Every variable, function and macro that has external link must have the unique prefix “KGL” or “kgl”.
  - A function must have a prefix KGL:
    ```c
    KGLForwardRealFFT();     Do not use underscores.
    ```
  - A macro must be in a upper-case and have a prefix KGL:
    ```c
    KGL_GITID
    ```
    But the same rule as a function may be applied for a function-like macro.
  - A structure, union or enumerate start with a lower-case letter:
    ```c
    kglCBCTemplateBank
    ```

- Every variable should have a different name from each other even if there is no overlap in the scope and/or the lifetime between any of two variables.
Name Space

- The rules of the prefix KGL/kgl are applied to names of functions, variables, etc. that have external link.

- This rule is not required for objects that can be accessed only in the file where they are declared.
  - It is recommended not to prefix “kgl” to distinguish local functions from global KAGALI functions
  - It is strongly recommended to declare such functions as **static** functions.
  - Access to static functions is restricted to the file where they are declared.

- The meaning of the “static” keyword here is different from that for static variables in a function.
% gcc -o test main.c src1.c src2.c
/tmp/ccrxW7lK.o: In function `func1':
src2.c:(.text+0x0): multiple definition of `func1'
/tmp/ccpbcw39.o:src1.c:(.text+0x0): first defined here
collect2: ld returned 1 exit status
// src1.c
#include <stdio.h>
int x = 0;
static void func1(void) {
    x++;
    return;
}
void sample1(void) {
    func1();
    printf("%d\n", x);
    return;
}

// src2.c
#include <stdio.h>
int x;
static void func1(void) {
    x = 500;
    return;
}
void sample2(void) {
    func1();
    printf("%d\n", x);
    return;
}

// main.c
void sample1(void); void sample2(void);
int main(void) {
    sample1();
    sample2();
    sample1();
    }

% gcc -o test main.c src1.c src2.c
% ./test
1
500
501
// src1.c
#include <stdio.h>
static int x = 0;
static void func1(void) {
    x++;
    return;
}
void sample1(void) {
    func1();
    printf("%d\n",x);
    return;
}

// src2.c
#include <stdio.h>
static int x;
static void func1(void) {
    x = 500;
    return;
}
void sample2(void) {
    func1();
    printf("%d\n",x);
    return;
}

// main.c
void sample1(void);   void sample2(void);
int main(void) {
    sample1();
    sample2();
    sample1();
}

% gcc -o test main.c src1.c src2.c
% ./test
1
500
2
Version Control

- We use “git” for the version control.
  - Itoh-san prepared the git server at Univ. of Tokyo.

- The KAGALI sources are located in the KAGALI git repository.

- Developers can pull/push the KAGALI sources from/to the repository.
  - It has not been open to the public yet.
Version Control

- The version identifier of KAGALI is embedded in the library via `KGLVersion.c` and `KGLVersion.h`.

  - When a developer modifies the source and commit it to git, `KGLVersion.h` will be updated automatically from the git-log.
  - It includes the date and the name of the latest committer as well as the commit hash of git.
Analysis tools of the sources

- To find bugs in the source codes, we need tools that can automatically detect various bugs.
- LAL provides the sophisticated mechanisms, including the error handling/reporting system and memory leak checking.
- We made a simpler mechanism of reporting errors during the execution of a program.
Memory leak checking

- For memory leak checking, users of LALsuit should call `LALMalloc()` and `LALFree()`, instead of `malloc()` and `free()`, as well as call `LALCheckMemoryLeaks()` before exiting a program.

- I feel it to be inconvenient for users who write various programs.

- We decided to use external applications.
cppcheck

- Cppcheck is a static analysis tool for C/C++ code. [http://cppcheck.sourceforge.net/](http://cppcheck.sourceforge.net/)
- Cppcheck primarily detects the types of bugs that the compilers normally do not detect, including:
  - Out of bounds checking
  - Memory leaks checking
  - Detect possible null pointer dereferences
  - Check for uninitialized variables
  - Warn if obsolete or unsafe functions are used
  - Warn about unused or redundant code
  - Detect various suspicious code indicating bugs
  - …
Nightly Build

- When the sources are modified, they will be checked by “cppcheck” nightly.
  - The automatic checking system was provided by Itoh-san under Jenkins on the KAGALI git server.
Memory leak checking

- Bugs including memory leaks that cppcheck can detect are limited.
- It must be detected **dynamically**; execute the program to check memory leak.

- **Valgrind** is an instrumentation framework for building dynamic analysis tools.
  - There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail.
**Required software**

- To use KAGALI, the following software packages are required:
  - GSL – The GNU Scientific Library (ver. 1.13 - 1.16)
    (Ver. 2.x is not supported.)
  - FFTW – The Fast Fourier Transform (ver. 3.2.0 or up)
  - FrameL – LIGO/Virgo Frame library (ver. 8.0.0 or up)
  - [optional]
    - MPI (OpenMPI, MPICH)
    - CBLAS (OpenBLAS, ATLAS)

- For a maintainer/developer,
  - GNU Autotools [autoconf/automake/libtools]
    (ver. 2.63 or up)
  - pkg-config (ver. 0.9.0 or up)
The present status of KAGALI

- The latest version of KAGALI is 0.4-alpha.
  - Korean group is developing the code for PE (MCMC) in a branch of 0.4-alpha.

- It is not open to the public, yet.
The present status of KAGALI

- The current source tree is
  - kagali
    - kglcommon
      - dataloading
      - detector
      - estimatepsd
    - cbc
    - hht
    - inference
    - waveform
  - kagali-apps
    - basic
    - cbc
    - nha
Code Review

- Code review is in progress; to check whether
  - functions work properly
  - programs are easy to understand, upon the coding style guide
  - optimal performance is obtained

- For code review
  - developers are requested to make independent programs with data files for examination.

- Developers are recommended to make sample programs to show how to use them.
Code Review

- Most of the present developers don’t have experience in making programs intended to be used by other persons.

- Most of the present programs are not satisfactory yet both in speed and easiness to understand.
Documents of KAGALI

- We are preparing documents of KAGALI.
  - **Tips on C Programming**
    - It is not the coding rules of KAGALI, but a collection of general hints useful to make programs that run fast and are easy to understand.
  - **Developer’s Manual of KAGALI**
    - It explains how to write KAGALI functions, sample programs, Makefile, etc., as well as how to download/upload the sources to the version control server.
  - **Quick Installation Manual of KAGALI**
    - for KAGALI users
  - **Reference Manual of KAGALI**
    - not yet
Summary

- Steady, not so quick, progress in development of KAGALI is being made.
- Code review was started, but not completed.
- Some parts of the library has been used for R&D of GW data analysis and will be used for analysis of iKAGRA data.
- It is still the alpha-version; we have to examine throughout the code.
- I hope that the beta-version can be open within this year.
- The first public version will be released by bKAGRA.