PROMISE

(PRecision OptiMISEd)

Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie - Paris 6
Paris, France
http://promise.lip6.fr

Contents

1 Introduction

PROMISE is a tool to auto-tune floating-point precision with Discrete Stochastic Arithmetic (DSA) [5], and in particular we use the CADNA library[1] [2, 3, 4], to check the accuracy requested by the user.
Two versions may be used. The first one runs the reference code and every tuned source code with CADNA. It is referred to as the `full stochastic` version. The second one runs only the reference code with CADNA and other executions are performed using standard floating-point types. It is referred to as the `stochastic reference` version.
PROMISE has been developed for C/C++ codes. Every variable is tunable. However, you can keep the definition types of some of them in single or in double precision.

2 Prerequisite

To use PROMISE you need:

- A C++ compiler
- Python
- CADNA for C/C++

The example codes described in this document have been run using g++ 4.9.2, Python version 2.7.9 and CADNA for C/C++ version 2.0.0.

3 Installation and test runs

3.1 Installation

First of all, you can indicate where CADNA is installed using the script `init.sh`.
The `src` folder contains the source codes needed to run PROMISE. Two versions of PROMISE exist and so you have to choose the full stochastic one (`make full`) or the reference stochastic one (`make ref`) according to the version you want to use. Then an executable called `Promise_compare` is generated. It is used to check the accuracy of a new configuration.

[1] can be downloaded from `http://cadna.lip6.fr`

3.2 Test runs

To run an example, go to the corresponding folder:

- arclength: An arclength computation
- MXM: A matrix multiplication
- rectangleMethod: rectangle method for the computation of integrals
- squareRoot: Babylonian method to approximate a square root
- CG: Conjugate Gradient method [1]
- SP: a scalar penta-diagonal solver [1]

In each folder `test_promise.py` compiles, executes and tests the source code using several functions:

- `compile`: compiles the code
- `test_significant_digit_high`: executes the code in double precision and checks the accuracy of the result
- `test_significant_digit`: executes a modified version of the code and checks the accuracy of the result.

You can modify the number of requested digits (`Digits`) in two functions: `test_significant_digit` and `test_significant_digit_high`.
The command to run an example is `../../src/promise_ref.py fileToTest` or `../../src/promise_full.py fileToTest` depending on the version of PROMISE you want to use.
The modified source code can be found in the `res` folder.

3.3 Summary

To run an example such as arclength with the full stochastic version:

```
$ ./init.sh
Enter the location of the CADNA library folder
~/cadna
$ cd src
$ make full
$ cd ../examples/arclength
$ ../../src/promise_full.py fileToTest
```

To run an example such as arclength with the stochastic reference version:

```
$ ./init.sh
Enter the location of the CADNA library folder
~/cadna
$ cd src
$ make ref
$ cd ../examples/arclength
$ ../../src/promise_ref.py fileToTest
```

4 Use PROMISE with your code

4.1 Example code

In this section we take the following code as an example:

```
#include <iostream>

int main ()
{
        double a = 3.14159;
        double b = 2.71828;
        double res;

        std::cout << "a   =  " << a << std::endl;
        std::cout << "b   =  " << b << std::endl;

        res = a * b;

        std::cout << "res  =  " << res << std::endl;
        return 0;
}
```

4.2 Preparation of the code

The `src/dump.h` file contains some functions to extract data and be able to use the reference stochastic version of PROMISE if you have tested your code with CADNA before and kept CADNA functions in it[2] and so should be included in your source code. Also as you may want to enable CADNA

[2] for example `cadna_init` or `strp`

at a specific point[3], you have to manually write "`cadna_init(0);`" at the requested place.
So we have the following modified code:

```
#include <iostream>
#include ''dump.h''

int main ()
{
        double a = 3.14159;
        double b = 2.71828;
        double res;

        cadna_init(0);

        std::cout << "a   =  " << a << std::endl;
        std::cout << "b   =  " << b << std::endl;

        res = a * b;

        std::cout << "res  =  " << res << std::endl;
        return 0;
}
```

4.3 Choose the result to test

Two functions may be used to extract the result:

- `dump(a)`: extracts the value of `a` in the `result.bin` file
- `dump(a, size)`: extracts the value of `a[i]` for `i` in $\{0, 1, ..., size - 1\}$ in the `result.bin` file

In our example, as we want to check the accuracy of `res` we will use the first one:

```
#include <iostream>
#include ''dump.h''

int main ()
```

[3] In particular, after an initialization

```
{
        double a = 3.14159;
        double b = 2.71828;
        double res;

        cadna_init(0);

        std::cout << "a   = " << a << std::endl;
        std::cout << "b   = " << b << std::endl;

        res = a * b;

        std::cout << "res = " << res << std::endl;
        dump(res);
        return 0;
}
```

4.4 To use the stochastic reference version of PROMISE

You should replace every variable type you want to test by `__PROMISE__`:

```
#include <iostream>
#include ``dump.h''

int main ()
{
        __PROMISE__ a = 3.14159;
        __PROMISE__ b = 2.71828;
        __PROMISE__ res;

        cadna_init(0);

        std::cout << "a   = " << a << std::endl;
        std::cout << "b   = " << b << std::endl;

        res = a * b;

        std::cout << "res = " << res << std::endl;
        dump(res);
        return 0;
```

}

You can create your own version of the test file (see part 4.7). You may run the `promise_ref.py` script as in 3.3.

4.5 To use the full stochastic version of PROMISE

The easiest way to run the full stochastic version is to call `cadnaizer_full` to create a C++ file with CADNA statements that will be used by PROMISE

```
../../src/cadnaizer_full source_file.cpp -o file_to_test.cpp
```

You can create your own version of the test file (see part 4.7). You may run the `promise_full.py` script as described in 3.3.

4.6 To get a final code with CADNA

With both versions of PROMISE you can use the option `-outCadna` to get a final code with CADNA. The possible command lines are in this case:

```
$ ../../src/promise_full.py fileToTest -outCadna
$ ../../src/promise_ref.py fileToTest -outCadna
```

4.7 Modify the test used

You can create your own test to validate a configuration. As explained before, the value `Digits` in the `test_promise.py` file is used as the accuracy requirement.
By default, the test uses an infinity norm in the `compare` function from the `Promise_Compare_ref.cc` or `Promise_Compare_full.cc` file. You may modify this norm to adapt it to your needs.

References

[1] Contributors of Center for Manycore Programming, Seoul. SNU NPB Suite, 2010.

[2] P. Eberhart, J. Brajard, P. Fortin, and F. Jézéquel. High performance numerical validation using stochastic arithmetic. *Reliable Computing*, 21(1):35–52, 2015.

[3] F. Jézéquel and J.-M. Chesneaux. CADNA: a library for estimating round-off error propagation. *Computer Physics Communications*, 178(12):933–955, 2008.

[4] J.-L. Lamotte, J.-M. Chesneaux, and F. Jézéquel. CADNA_C: A version of CADNA for use with C or C++ programs. *Computer Physics Communications*, 181(11):1925–1926, 2010.

[5] J. Vignes. Discrete Stochastic Arithmetic for validating results of numerical software. *Numerical Algorithms*, 37(1–4):377–390, December 2004.