# CADNA demo

Fabienne Jézéquel

Sorbonne Université, Laboratoire d'Informatique de Paris 6 (LIP6), France

Workshop on Large-scale Parallel Numerical Computing Technology

RIKEN Center for Computational Science, Kobe, Japan, 6-8 June 2019

# CADNA installation

CADNA can be freely downloaded from `http://cadna.lip6.fr`

```
gunzip cadna_c-3.1.3.tar.gz
tar -xvf cadna_c-3.1.3.tar
cd cadna_c-3.1.3
./configure --prefix=‘pwd‘ --enable-fortran
make install
```

Automatic detection of OpenMP, MPI. Up to 10 libraries can be created.

Ex: libcadnaC.a, libcadnaCdebug.a, libcadnaMPICforOpenMP.a, libcadnaMPICdebugforOpenMP.a, libcadnaMPIFortran.a,...

- optimized versions: inlining, -O3
- debug versions: no inlining, -O0, -g

Successfully tested with GNU (gcc, gfortran), IBM (xlc, xlf), Intel (icpc, ifort), LLVM (clang, xlflang), PGI (pgcc, pgfortran) compilers.

# Outline

# Example 1 [S.M. Rump, 1988]

$P = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$

with $x = 77617$ and $y = 33096$

Exact result : $P \approx$-0.827396059946821368141165095479816292

Execution without/with CADNA

# Example 2

The roots of the following second order equation are computed:

$$0.3x^2 - 2.1x + 3.675 = 0.$$

The exact values are:

- Discriminant $d = 0$
- $x_1 = x_2 = 3.5$

Execution without/with CADNA:

- without CADNA:
  wrong branching ⇒ the result is false
- with CADNA:
  $if\ (d == 0.)$ is satisfied if $d$ is numerical noise

# Example 3

The determinant of Hilbert's matrix of size 11 defined by

$$a_{i,j} = 1/(i+j-1)$$

is computed without pivoting strategy.

After triangularization, the determinant is the product of the diagonal elements.

Execution without/with CADNA

# Example 4 [J.-M. Muller, 1987]

The 25 first iterations of the following recurrent sequence are computed:

$$U_{n+1} = 111 - 1130/U_n + 3000/(U_n * U_{n-1})$$

with $U_0 = 5.5$ and $U_1 = 61/11$.

The exact limit is 6.

Execution without/with CADNA

With CADNA, instabilities related to DSA self-validation are detected.
Then, the accuracy estimation is not reliable.

# Example 5

A root of the polynomial

$$f(x) = 1.47x^3 + 1.19x^2 - 1.83x + 0.45$$

is computed by Newton's method. The sequence is initialized with $x = 0.5$.
The iterative algorithm

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

is stopped by the criterion

$$|x_n - x_{n-1}| < 10^{-12}.$$

💻 Execution without/with CADNA

# Example 5

Without CADNA

- stopping criterion `fabs(x-y)<1.e-12`

  x( 35) = +4.285714252078272e-01
  x( 36) = +4.285714252078272e-01

  The stopping criterion is satisfied by chance.

With CADNA

- stopping criterion `fabs(x-y)<1.e-12`

  x(100) = 0.4285714E+000
  x(101) = 0.4285714E+000

  if `x-y` is numerical noise, the test is not satisfied
  ⇒ maximum number of iterations

  Because of instable divisions detected by CADNA, a multiple root is suspected.

# Example 5

With CADNA

- stopping criterion `fabs(x-y)<=1.e-12` or `x==y`

  x( 23) = 0.428571437E+000
  x( 24) = 0.42857143E+000

  if `x-y` is numerical noise, the test is satisfied.

- we simplify the faction, and so compte a simple root.

  stopping criterion `x==y`

  x( 45) = 0.428571428571430E+000
  x( 46) = 0.428571428571429E+000

# Example 6

A linear system of size 4 is solved using Gaussian elimination with partial pivoting.

Execution without/with CADNA

- Without CADNA

  when `i=2`, `a[2][2]` is 4864.
  But that value has actually no correct digits (associate exact value: 0).
  `a[2][2]` is chosen as the pivot and leads to round-off errors in the subsequent computation.

- With CADNA

  One can observe that `a[2][2]` has no correct digits.
  The test `fabsf(a[j][i])>pmax` fails.
  `a[3][2]`, that is computed accurately, is chosen as the pivot.

# Example 7

## Example created on purpose to make CADNA fail

We compute several times:

```
x=6.83561e+5;  y=6.83560e+5;  z=1.00000000007;
r = ((z-x)+y) + ((z-y)+x-2);
```

Exact result: $1.4\,10^{-10}$

Execution without/with CADNA

Without CADNA:
using IEEE double precision with rounding to nearest

```
r=2.32830643653870E-10
```

With CADNA:
we perform close evaluations: $((z-x)+y)$ and $((z-y)+x-2)$.
If the same rounding mode is chosen for both parts, the final result appears as exact but it is wrong.
1 case in 4 CADNA provides `0.116415321826935E-009`, otherwise `@.0`

# Example with OpenMP [Eberhart et al., 2016]

The sum S of an array $A$ of size $2n$ with $n = 10^6$ is computed in single precision:

```
#pragma omp parallel for
for (i=0;i<2*n;i=i+2) {
     A[i+1]=(float)i+1.;
     A[i]=-(float)i;
   }
S=0.;
#pragma omp parallel for reduction(+:S) schedule(static,1)
for (i=0;i<2*n;i++)
     S=S+A[i];
```

Exact result: $S = 10^6$

Execution without/with CADNA

# Two scheduling options

| # threads | (static) without CADNA | (static) with CADNA | (static,1) without CADNA | (static,1) with CADNA |
|---|---|---|---|---|
| 1 | 1.000000E+06 | 1.000000E+06 | 1.000000E+06 | 1.000000E+06 |
| 2 | 1.000000E+06 | 1.000000E+06 | 1.966080E+06 | @.0 |
| 3 | 1.000000E+06 | 1.000000E+06 | 1.000000E+06 | 1.000000E+06 |
| 32 | 1.000000E+06 | 1.000000E+06 | 1.892352E+06 | @.0 |
| 64 | 1.000000E+06 | 1.000000E+06 | 1.787904E+06 | @.0 |
| 128 | 1.000000E+06 | 1.000000E+06 | 1.609728E+06 | @.0 |
| 239 | 1.000000E+06 | 1.000000E+06 | 1.000000E+06 | 1.000000E+06 |
| 240 | 1.000000E+06 | 1.000000E+06 | 1.617920E+05 | @.0 |

- (static): the loop is divided into chunks of size $2n/T$ where $T$ is the number of threads.
  Each thread has in charge contiguous elements of A and alternatively sums positive and negative values: all results are accurate.
- (static,1): the chunk size is 1.
  - odd number of threads: correct result
  - even number of threads: the result has no correct digits

# With (static,1) and an even number of threads

- With 2 threads:

| $A:$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

thread 0 (light cells: 0, 2, 4, 6, 8, 10) thread 1 (dark cells: 1, 3, 5, 7, 9, 11)

thread 0 computes $P(n) = \sum_{i=0}^{n-1}(-2i) = -n^2 + n$
and thread 1 $Q(n) = \sum_{i=0}^{n-1}(2i+1) = n^2$.

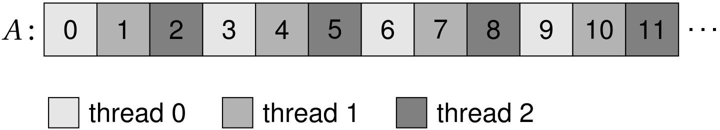
For $k = 2, \ldots, n$,
$P(k) = P(k-1) - (2k-2) = -((k-1)^2 + k - 1) - (2k-2)$
and $Q(k) = Q(k-1) + 2k - 1 = (k-1)^2 + 2k - 1$.

The computation of $P(k)$ and $Q(k)$ involves values with different orders of magnitude and generates round-off errors for $k$ sufficiently high.

- With an even number of threads $> 2$: same phenomenon.
  The reduction involves inaccurate results with close absolute values and different signs and thus provides numerical noise.

# With (static,1) and an odd number of threads

For example, with 3 threads:




Each thread has in charge positive and negative elements of A.

No cancellation occurs and all results are accurately computed.

# Example with MPI [S.M. Rump, 1983]

We compute using 4 MPI processes $P = 9x^4 - y^4 + 2y^2$ with $x = 10864$ and $y = 18817$.

Processes 1, 2 and 3:

- compute respectively $9x^4$, $-y^4$ and $2y^2$
- send their local result to process 0.

Process 0 receives and sums the results.

Exact result : $P = 1$.

Execution without/with CADNA

```
mpirun -np 4 exampleMPI1
mpirun -np 4 exampleMPI1_cad
```

# Example with MPI and OpenMP [S.M. Rump, 1983]

We compute using 4 MPI processes $P = 9x^4 - y^4 + 2y^2$ with $x = 10864$ and $y = 18817$.

Processes 1, 2 and 3:

- compute respectively $9x^4$, $-y^4$ and $2y^2$ in parallel using OpenMP
- send their local result to process 0.

Process 0 receives and sums the results.

Exact result : $P = 1$.

Execution without/with CADNA

```
mpirun -np 4 exampleMPI_OMP1
mpirun -np 4 exampleMPI_OMP1_cad
```

# Outline

- version of CADNA written in Fortran
- CADNA in C can be used in Fortran codes thanks to Fortran/C binding

Example: addition of two double_st variables (in srcFortran)

in cadna_add.f90:

```
interface operator(+)
  module procedure add_double_st_double_st
end interface operator(+)
interface
  pure function cpp_add_double_st_double_st(a, b) bind(C)
    import double_st
    type(double_st), intent(in) :: a
    type(double_st), intent(in) :: b
    type(double_st) cpp_add_double_st_double_st
  end function cpp_add_double_st_double_st
end interface
```

in cadna_add_binding.cc:

```
 double_st cpp_add_double_st_double_st( double_st &a,  double_st &b )
{ return a+b; }
```

+ specific Fortran sources for overloading of array functions (MATMUL, SUM, PRODUCT,...)

💻 Executions without/with CADNA in examplesFortran

# Outline

# The CADTRACE tool

CADTRACE can be freely downloaded from `http://cadna.lip6.fr`

## Installation

```
gunzip cadtrace-2.2.tar.gz
tar -xvf  cadtrace-2.2.tar
cd cadtrace-2.2
./configure --prefix=‘pwd‘
make install
```

Use `gdb` with the `gdb_c.in` file provided in the `extra-files` directory and generate a `gdb.out` file.

Example with a C program

in the `examplesC` directory:

```
gdb ex5_cad<gdb_c.in>gdb.out
```

To obtain a detailed list of instabilities:

```
cadtrace_gcc gdb.out
```

Example with a Fortran program

in the `examplesFortran` directory:

```
gdb ex5_cad<gdb_c.in>gdb.out
```

To obtain a detailed list of instabilities:

```
cadtrace_gcc gdb.out
```

You can also specify the number of function calls that generate each instability. For instance, to get 3 levels of function calls:

```
cadtrace_gcc -n 3 gdb.out
```

Remark: the `cadna_enable`, `cadna_disable` functions may help for numerical debugging.

Thanks to Jean-Marie Chesneaux, Julien Brajard, Romuald Carpentier, Patrick Corde, Pacôme Eberhart, Pierre Fortin, Jean-Luc Lamotte, ...

Thanks to Jean-Marie Chesneaux, Julien Brajard, Romuald Carpentier, Patrick Corde, Pacôme Eberhart, Pierre Fortin, Jean-Luc Lamotte, ...

Thank you for your attention!