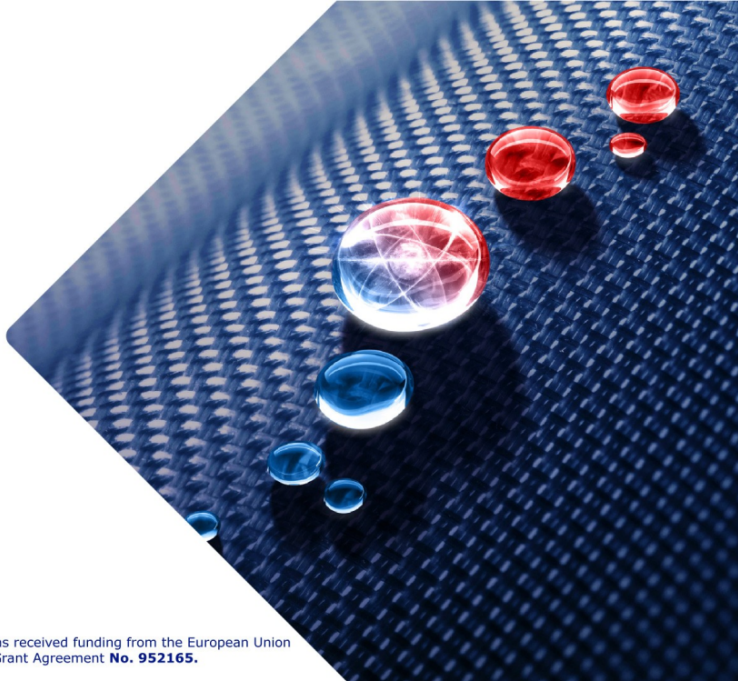
Targeting Real chemical accuracy at the EXascale

A Mixed-precision exploration of CHAMP

François Coppens

17/05/2021

Université de Versailles Saint-Quentin-en-Yvelines



Targeting Real Chemical Accuracy at the Exascale project has received funding from the European Union Horizoon 2020 research and innovation programme under Grant Agreement **No. 952165.**

Quick reminders

Language/compiler support

- C/C++/`clang`: LLVM versions 4.0 – 15.0
- Fortran/`flang`: LLVM versions $\leq$ 7.0

SSH connection to OLYMPE

```
> ssh -X <username>@olympe.calmip.univ-toulouse.fr
```

Minimal environment

1. `> export MODULEPATH=${MODULEPATH}:/usr/local/trex/modulefiles`
2. `> module load verificarlo`

Verificarlo Singularity container

```
> sing-verificarlo
```

Verificarlo is a LLVM-based compiler extension. To this end Verificarlo offers compiler wrappers for the languages:

C	`verificarlo-c`
C++	`verificarlo-c++`
Fortran	`verificarlo-f`

To compile and link a single source file

```
verificarlo-<lang> source-file.<lang> -o <executable name>
```

Or compile and link several files

```
verificarlo-<lang> -c source-file1.<lang> -o source-file1.o
verificarlo-<lang> -c source-file2.<lang> -o source-file2.o
...
...

verificarlo-<lang> source-file1.o source-file2.o ... -o <executable>
```

To use the Verificarlo compiler and linker in Makefile or CMake projects we recommend setting the environment variables `CC`, `CXX` and `FC` for respectively the C, C++ and Fortran compiler.

These are usually respected by GNU Make and Kitware CMake, unless explicitly overridden by by the user. To do this execute the following commands in the terminal

- `export CC=verificarlo-c`
- `export CXX=verificarlo-c++`
- `export FC=verificarlo-f`

Use OpenMPI wrapper scripts to include OpenMPI headers and libraries. Make sure to point these wrappers to the appropriate Verificarlo compiler wrappers by exporting the following environment variables

- `export CC=mpicc`
- `export CXX=mpic++`
- `export FC=mpifort`
- `export OMPI_C=verificarlo-c`
- `export OMPI_CXX=verificarlo-c++`
- `export OMPI_FC=verificarlo-f`

This will ensure the compiler wrappers are chained in the correct order.

To use the Delta-Debug feature in Verificarlo the extra flag **`--ddebug`** needs to be passed to the compiler directly or as `CFLAGS/CXXFLAGS/FCFLAGS`

```
verificarlo-{c|c++|f} --ddebug \
-c source-file.<lang> \
-o source-file.o
```

or as

```
CFLAGS/CXXFLAGS/FCFLAGS += --ddebug
```

Mixed precision exploration in CHAMP using Delta-Debug

- Delta-Debug in Verificarlo
  - Configurations are the sets of floating-point instructions.
  - A bug is a numerical instability.
- Find unstable instructions for rounding / cancellations.
- Find instructions that can be run in lower precision.

Step	Instructions with MCA noise	Numerically Stable
1	1 2 3 4	stable
2	. . . . 5 6 7 8	unstable
3	. . . . 5 6 . .	stable
4	. . . . . 7 8	unstable
5	. . . . . 7 .	unstable
Result (ddmin)	. . . . . 7 .	

Setting up the Delta-Debug helper scripts

1. Chose an observable: in this case it's the **total energy** and **one of the forces**
2. Prepare `ddRun`: script that runs the code and extracts the observable to monitor from the output
3. Prepare `ddCmp`: script that sets the **precision threshold** from the reference value and returns:
   - `true/pass`: if precision of current value does **NOT EXCEED** maximum deviation
   - `false/fail`: if precision of current value **EXCEEDS** the maximum deviation
4. Prepare a small `Makefile`: to automate setting Verificarlo **backend**, **precision**, **# of ddebug runs** and `vfc_ddebug` invocation

```
OUTDIR=$1

BIN=./vfcdd_champ
INP=input.inp

for i in $(seq 1 1)
do
  $BIN -i $INP -o /tmp/res.dat 2> ${OUTDIR}/ddrun.error
  awk '/total E/{print$4}' /tmp/res.dat > $OUTDIR/res$i.dat
  rm /tmp/res.dat
done
```

```
import sys
import glob
import os
import numpy as np
import math

PRECISION_THRESHOLD = 5 # number of DECIMAL digits
REFDIR  = sys.argv[1]
CURRDIR = sys.argv[2]
res = np.zeros(shape=0)
```

```
def read_output(DIR, RES):
    resList = glob.glob(os.path.join(DIR, 'res*.dat'))
    resList += glob.glob(os.path.join(REFDIR, 'res*.dat'))
    for resFile in resList:
        with open(resFile) as f:
            RES = np.append(RES, float(f.read()))
    return RES
```

```
res = read_output(CURRDIR, Res)

s = math.log2(abs((res[0]-res[1])/res[1]))/math.log2(10)
          # see next slide

with open("{}/res.stat".format(CURRDIR), 'w') as f:
    print(f"Stat. file loc = {CURRDIR}/res.stat")
    f.write("s = {}\n".format(s))

sys.exit(1 if s < PRECISION_THRESHOLD else 0)
```

Precision in number of bits

$$s_2 = -\log_2 \left| \frac{x_{\mathrm{VPREC}} - x_{\mathrm{IEEE}}}{x_{\mathrm{IEEE}}} \right|$$

Precision in number decimals

$$s_{10} = \frac{s_2}{\log_2(10)}$$

```
NRUNS=1
PRECISION=24
BACKEND="libinterflop_vprec.so --precision-binary64=${PRECISION}"

dd: vfcdd_champ
  rm -rvf dd.line/
  INTERFLOP_DD_NRUNS=${NRUNS} VFC_BACKENDS=${BACKEND} \
  vfc_ddebug ddRun_vp ddCmp_vp

dderrors: dd.line/rddmin-cmp/dd.line.exclude
  bash -c "vim -q <(./vfc_dderrors.py ./vmc $<)"
```

After running `make dd` you should see output like

```
INTERFLOP_DD_NRUNS=1 VFC_BACKENDS="libinterflop_vprec.so \
   --precision-binary64=17" vfc_ddebug ddRun_vp ddCmp_vp
dd.line/8a391a01211abd91e3d54cfeca2897f4 --( run )-> FAIL(0)
dd.line/8a391a01211abd91e3d54cfeca2897f4 --(cache) -> FAIL
dd.line/525c227b98a0ae2e100829a4100dbcf4 --( run )-> FAIL(0)
dd.line/4ebd0a8a27fa25253644536b5ea1cbe8 --( run )-> FAIL(0)
dd.line/8a8069fe0ee2b665eab4c7034fba118d --( run )-> PASS(+1->1)
dd.line/b434b51d52907341f6f05a9e4128d8e1 --( run )-> FAIL(0)
dd.line/53769469e1628356f7b2aa219c9df18a --( run )-> FAIL(0)
dd.line/d2d274f8db7675ba550c01ce6979c32a --( run )-> PASS(+1->1)
dd.line/93e0258c4d272730e115c64951366207 --( run )-> FAIL(0)
dd.line/c7d60fc315f6b8e0ec55cb39c6a4edd1 --( run )-> PASS(+1->1)
dd.line/91384f0e34618621669fdd58e04583c3 --( run )-> FAIL(0)
ddmin0 (0x0000000000521ced: splfit_ at splfit.f:47):
```

```
drwxr-xr-x. coppens p22064 04bb7a6780419ee429cd806fe66cce66
drwxr-xr-x. coppens p22064 05928b62472d3723566e3b4d193bcb2c
drwxr-xr-x. coppens p22064 0598f288b121391b3cd8f9f2e948c8d2
drwxr-xr-x. coppens p22064 d914463b3d0055e77e0128b74ceb5291
lrwxrwxrwx. coppens p22064 ddmin0 -> 05928b62472d3723566e3b4d193bcb2c
lrwxrwxrwx. coppens p22064 ddmin1 -> d914463b3d0055e77e0128b74ceb5291
drwxr-xr-x. coppens p22064 dee78da1159ba7d45cfd5809f9799e22
drwxr-xr-x. coppens p22064 df02513b130ff37c200ebf49f7aa22a7
drwxr-xr-x. coppens p22064 fd5e7760f41617300b4e5b49b6a55843
drwxr-xr-x. coppens p22064 ff680b1e38e2818b5e9f67029f123644
lrwxrwxrwx. coppens p22064 rddmin-cmp -> df02513b130ff37c200ebf49f7aa22a7
drwxr-xr-x. coppens p22064 ref
```

```
ff680b1e38e2818b5e9f67029f123644
|-- dd.line.exclude
|-- dd.line.include
`-- dd.run1
    |-- dd.compare.err
    |-- dd.compare.out
    |-- dd.run.err
    |-- ddrun.error
    |-- dd.run.out
    |-- res1.dat
    |-- res.stat
    `-- returnVal
```

```
ref
|-- checkRef.err
|-- checkRef.out
|-- dd.err
|-- dd.line
|-- dd.line.%%p
|-- dd.out
|-- ddrun.error
|-- res1.dat
`-- res.stat
```

When done, minimal set can be found in `$PWD/dd.line/rddmin-cmp`. It contains the files

- **`dd.line.exclude`:** contains the functions/subroutines that break at the chosen precision.
- **`dd.line.include`:** contains the functions/subroutines that can be changed to the lower precision without affecting the precision of the total energy

Table: CHAMP/VMC Butadiene CIPSI. Investigated precision on TOTAL ENERGY for all functions.

Function name	Required precision (VPREC)
`splfit`	Double
`ALL OTHERS`	Single

Conclusion: When only interested in the energy we can run most of CHAMP at single precision, potentially gaining a significant speedup.

CHAMP results for forces

Table 2: CHAMP/VMC Butadiene CIPSI. Investigated precision on FORCES for functions of runtime >= 5% of total run-time.

	Time spent (%)		
Function name	500 dets	15k dets	Required precision (VPREC)
`orbitals`	22.02	5.71	Double
`nonloc`	11.7	3.15	
> `orbitals_quad:395`	7		Single
`orbitalse`	5.56	3.36	Not called in VFC-DD
`optjas_deloc`	4.66	16.45	Singel
`splfit`	4		Double
`multideterminante_grad`	3.62	2.48	
`multideterminante`	3.59	14.86	Double
`multideterminant_hpsi`	3.56	3.49	
`n0_inc`	2.6		
`basis_fns_vgl`	2.48		
`basis_fnse_v`	2.43		
`optorb_compute`	2.08		
`compute_ymat`		12.8	Double
`detsav`		4.66	Not called in VFC-DD
`__powr8i4`	7.16	1.88	
`__libm_log_l9`	2.43	0.58	

Conclusion: sometimes life is more complicated

MPI `flang-7` is not able to assign multiple file descriptors to the same file $\longrightarrow$ prevents from running more than one MPI process!

MORE PROBLEMS TO BE ADDED