Study of discrepancies in CHARMM across heterogeneous platforms

Trilce Estrada
Global Computing Lab (GCLab)
University of Delaware
CHARMM: Chemistry at HARvard
Macromolecular Mechanics

- Is a widely used molecular simulation program with broad application to many-particle systems
- Provides a large suite of computational tools
- Can be used with various energy functions and models
  - Mixed quantum mechanical-molecular mechanical force fields
  - All-atom classical potentials with explicit solvent and various boundary conditions
  - Implicit solvent and membrane models
- Has been ported to numerous platforms in both serial and parallel architectures
Motivation: docking@home

BOINC clients

Processor: Intel
OS: Windows
Charmm.exe

Processor: AMD
OS: Windows
Charmm.exe

Processor: Intel
OS: Windows
Charmm.exe
Motivation: docking@home

BOINC clients

- Processor: **Intel**
  - OS: **Windows**
  - Charmm.exe

- Processor: **AMD**
  - OS: **Windows**
  - Charmm.exe

- Processor: **Intel**
  - OS: **Windows**
  - Charmm.exe

BOINC Server
Where do we find discrepancies?

- From 1842 results, 530 were invalid
  - For 270 the primary cause was a different operating system
  - For 226 the primary cause was a different processor vendor
  - For 34 the primary cause was not determined

- Different OS produce discrepant results. This behavior is expected because they run a different charmm compilation.

- Different processors with the same OS produce different results even if running the same charmm compilation.
3 Hypotheses
Methodology
Testing of 2 hypotheses
Discussion over the third hypothesis
Possible directions of this work
Hypotheses

- **Compilation:** the compilation of charmm enables aggressive optimizations that do not conform to the IEEE 754 floating point arithmetic standard

- **Programming:** the programming style leads to errors due to very small and very large values

- **Hardware:** architecture dependent issues. Processors of the IA32 architecture feature a floating-point unit known as x87 [Int 2005]
Methodology

1.- Select and test their IEEE 754 compatibility (paranoia)

- Paranoia is a program, originally written in BASIC by Professor W M Kahan, which tests the properties of the floating point arithmetic used on a computer
Methodology

1.- Select and test their IEEE 754 compatibility (paranoia)

2.- Compile charmm using different flags

3.- Execute the different compilations of charmm on the selected machines

4.- Determine which compilations produce mismatching results.

5.- Analyze the sections of the code that produce the discrepancies
Machines at the GCL:

- **Italy**: Optiplex 740 : AMD Athlon 64 X2 Dual Core 6000+ 3GHz
- **Finland**: Celeron 440 2GHz
- **Sweden**: Core2 Duo E6850 3GHz
- **Tanzania**: Intel(R) Pentium(R) 4 CPU 3GHz

Paranoia

- The tests with Paranoia were consistent across platforms, so, no evident problem with the IEEE 754 standard
The compilation of charmm enables aggressive optimizations that do not conform to the IEEE 754 floating point arithmetic standard

- If the compilation is the issue, then the problem can be solved by selecting the appropriate compiler flags
- Compiler: Intel Fortran v 10.1
The standard defines formats for representing:

- Floating-point numbers (including negative zero and denormal numbers)
- Special values (infinities and NaNs)
- Floating-point operations that operate on these values
- Four rounding modes
- Five exceptions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>n ÷ ±Infinity</td>
<td>0</td>
</tr>
<tr>
<td>±Infinity × ±Infinity</td>
<td>±Infinity</td>
</tr>
<tr>
<td>±nonzero ÷ 0</td>
<td>±Infinity</td>
</tr>
<tr>
<td>Infinity + Infinity</td>
<td>Infinity</td>
</tr>
<tr>
<td>±0 ÷ ±0</td>
<td>NaN</td>
</tr>
<tr>
<td>Infinity - Infinity</td>
<td>NaN</td>
</tr>
<tr>
<td>±Infinity ÷ ±Infinity</td>
<td>NaN</td>
</tr>
<tr>
<td>±Infinity × 0</td>
<td>NaN</td>
</tr>
</tbody>
</table>
Selection of compiler flags

Floating Point
-------------
- [no-]ftz enable/disable flush denormal results to zero
- [no-]prefetch enable(DEFAULT)/disable prefetch insertion
- [no]recursive compile all procedures for possible recursive execution

- fp-model <name> enable <name> floating point model variation
  - [no-]except enable/disable floating point semantics
  - fast[=1|2] enables more aggressive floating point optimizations
  - precise allows value-safe optimizations
  - source enables intermediates in source precision
  - strict enables -fp-model precise -fp-model except, disables
    contractions, enables property to allow for
    modification of the floating point environment

- fp-speculation <mode>
  enable floatation point speculations with the following <mode>
  conditions:
  - fast - speculate floating point operations (DEFAULT)
  - safe - speculate only when safe
  - strict - same as off
  - off - disables speculation of floating-point operations
Selection of compiler flags

-mp  maintain floating point precision (disables some optimizations)

-mp1  improve floating-point precision (speed impact is less than -mp)

-m[no-]ieee-fp
  same as -mp

-[no]fltconsistency
  specify that improved floating-point consistency should be used

-fpe{0|1|3}
  specifies behavior on floating point exceptions

-[no-]prec-div
  improve precision of floating-point divides (some speed impact)

-[no-]prec-sqrt
  determine if certain square root optimizations are enabled

-[no-]fp-port round fp results at assignments & casts (some speed impact)

-fp-stack-check enable fp stack checking after every function/procedure call

-pc32  set internal FPU precision to 24 bit significand

-pc64  set internal FPU precision to 53 bit significand

-pc80  set internal FPU precision to 64 bit significand (DEFAULT)

-rcd  rounding mode to enable fast float-to-int conversions

-rounding-mode chopped
  set internal FPU rounding control to truncate
FD:
- FFLAGS = -DPIC -fPIC -O3 -c -W0 -fexceptions -pc64 -axP -xW

F1:
- FFLAGS = -DPIC -fPIC -O3 -c -W0 -fexceptions -pc64

F2:
- FFLAGS = -DPIC -fPIC -O3 -c -W0 -fexceptions -pc64 -fp-model strict -fp-speculation off -mp -prec-sqrt -prec-div

F3:
- FFLAGS = -DPIC -fPIC -O0 -c -W0 -fexceptions -pc64 -fp-model strict -fp-speculation off -mp -prec-sqrt -prec-div
**Executables**

- **FD**: Charm32_linux64_intel_fd.exe
  - Size: 9,590,550 bytes
- **F1**: Charm32_linux64_intel_f1.exe
  - Size: 9,430,707 bytes
- **F2**: Charm32_linux64_intel_f2.exe
  - Size: 9,499,890 bytes
- **F3**: Charm32_linux64_intel_f3.exe
  - Size: 9,468,428 bytes
<table>
<thead>
<tr>
<th>Runs</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>FD</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD (5000min 20000vv)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1c70 (10x10)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1d4h (10x10)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>7upj (10x10)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1tng - 367835 (80x80)</td>
<td>✓</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>1tng - 298754 (80x80)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>1tng - 345897 (80x80)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Discrepancies in 1tng - 367835

- FD: All results are equal across machines
- F1: All results are equal across machines
- F2: Italy (AMD) has different results than the rest of the machines
- F3: Italy (AMD) has different results than the rest of the machines
  - prec-sqrt, prec-div

Testing of hypothesis 1 was not exhaustive enough to prove or discard this hypothesis
Hypothesis (2)

- Programming style leads to errors due to very small and very large values
  - Cancellation
  - Flushing of subnormal numbers
  - Addition or subtraction of numbers with very different magnitudes
Identification of discrepancies

1tng - 367835 (80x80)

| 3.63697  -10.2858 1TNG 5 30 1.757627E+06 | 3.63697  -10.2858 1TNG 5 30 1.757627E+06 |
| 6.819945E-02  -23.9449 1TNG 5 31 1.757627E+06 | 6.819945E-02  -23.9449 1TNG 5 31 1.757627E+06 |
| 3.18474  -9.08524 1TNG 5 32 1.757627E+06 | 3.18474  -9.08524 1TNG 5 32 1.757627E+06 |
| 4.13359  -11.0509 1TNG 5 33 1.757627E+06 | 4.13359  -11.0509 1TNG 5 33 1.757627E+06 |
| 3.63686  -10.2797 1TNG 5 34 1.757627E+06 | 3.63686  -10.2797 1TNG 5 34 1.757627E+06 |
| 0.632734  -24.4924 1TNG 5 35 1.757627E+06 | 0.632734  -24.4924 1TNG 5 35 1.757627E+06 |
| **3.16068  -8.99447 1TNG 5 36 1.757627E+06** | **3.16061  -9.05077 1TNG 5 36 1.757627E+06** |

Conformation 5, Rotation 36

After 184 trials
SEED FOR RANDOM NUMBER GENERATOR IS 609529063  
GAUSSIAN OPTION IS 1  
VELOCITIES ASSIGNED AT TEMPERATURE = 414.8366

DETAILS ABOUT CENTRE OF MASS
POSITION : 4.412355 -8.3975855
VELOCITY : 2.41908007E-02 -6.70166963E-02
ANGULAR MOMENTUM : 8.9704414 4.3729530
KINETIC ENERGY : 0.30133139

DETAILS ABOUT CENTRE OF MASS
POSITION : 4.412355 -8.3975855
VELOCITY : 2.02191353E-16 2.63432240E-16 -
ANGULAR MOMENTUM : -3.98583953E-14 -3.60069640E-14 -
KINETIC ENERGY : 2.16447737E-30

Dyna DYN: Step Time TOTEN TOTKE E Dyna DYN: Step Time TOTEN TOTKE E
Dyna PROP:  GRMS HFCtote HFCKe E Dyna PROP:  GRMS HFCtote HFCKe E
Dyna INTERN: BONds ANGLeS UREy-b DIHe Dyna INTERN: BONds ANGLeS UREy-b DIHe
Dyna EXTERN: VDWaals ELEC HECOnds Dyna EXTERN: VDWaals ELEC HECOnds
Dyna GRID: GrvDw GrElec Dyna GRID: GrvDw GrElec
Dyna PRESs: VIRE VIRI PRESse E Dyna PRESs: VIRE VIRI PRESse E
IF(.NOT.QALWRT(4)) THEN
    YCM=ZERO
    ZCM=ZERO
ENDIF
    IF(.NOT.QALWRT(5)) THEN
        XCM=ZERO
        ZCM=ZERO
    ENDIF
    IF(.NOT.QALWRT(6)) THEN
        XCM=ZERO
        YCM=ZERO
    ENDIF
ENDIF
C
AXCM=AXCM-(YCM**2ZCM-2CM*VYCM)/TMAS
AYCM=AYCM-(ZCM**2XCM-XCM*VZCM)/TMAS
AZCM=AZCM-(XCM**2VXCM-VCM*VXCM)/TMAS
XCM=XCM/TMAS
YCM=YCM/TMAS
ZCM=ZCM/TMAS
VXCM=VXCM/PMAS
VYCM=VYCM/PMAS
VZCM=VZCM/PMAS
EKCM=(VXCM**2+VYCM**2+VZCM**2)
EKCM=EKCM*TMAS/2M
C.##IF TSM
C.##ENDIF
IF(PRNLEV.GE.2) WRITE(OUTU,101) XCM, YCM, ZCM, VXCM, VYCM, VZCM,
  2                   AXCM, AYCM, AZCM, EKCM,
  101 FORMAT(/5X,'DETAILS ABOUT CENTRE OF MASS',/,
  2       5X,'POSITION'        :',1P,3G17.8,/,,
  3       5X,'VELOCITY'        :',1P,3G17.8,/,,
  4       5X,'ANGULAR MOMENTUM':',1P,3G17.8,/,,
  5       5X,'KINETIC ENERGY'  :',1P,G17.8)
DO I=1,NATOM
   IF(IMOVE(I).EQ.0) THEN
      AMASSI=AMASS(I)
      TMASS=TMASS+AMASSI
      XI=X(I)
      YI=Y(I)
      ZI=Z(I)
      VXI=VX(I)
      VYI=VY(I)
      VZI=VZ(I)
      VXCM=VXCM+VXI*AMASSI
      VYCM=VYCM+VYI*AMASSI
      VZCM=VZCM+VZI*AMASSI
      XCM=XCM+XI*AMASSI
      YCM=YCM+YI*AMASSI
      ZCM=ZCM+ZI*AMASSI
      AXCM=AXCM+(YI*VZI-ZI*VYI)*AMASSI
      AYCM=AYCM+(ZI*VXI-XI*VZI)*AMASSI
      AZCM=AZCM+(XI*VYI-YI*VXI)*AMASSI
   ENDIF
ENDDO

Tracking of variables
Finding discrepancies

The very first discrepancy was found at:

- **VXI = -1.495125283963315E-003**
  - Exponent: 01111110101
  - Significand: 1.1000011111111000000010100011
    001010110011000101100

- **VXI = -1.495125283963314E-003**
  - Exponent: 01111110101
  - Significand: 1.1000011111111000000010100011
    001010110011000101100
Programming issues

- VXI is updated in 24 places across 7 files
  - Find the operations involved in the updating of VXI and determine the range of values in those operations

```fortran
DO I=1, NATOM
  IF(IMOVE(I).EQ.0) THEN
    SD=BOLTZ/AMASS(I)
    SD=SQR2(SD)
    CALL GAUSSI(ZERO, SD, VEL, ISEED, IASVEL)
    VX(I)=VEL
    CALL GAUSSI(ZERO, SD, VEL, ISEED, IASVEL)
    VY(I)=VEL
    CALL GAUSSI(ZERO, SD, VEL, ISEED, IASVEL)
    VZ(I)=VEL
  ELSE
    VX(I)=ZERO
    VY(I)=ZERO
    VZ(I)=ZERO
  ENDIF
  IF(PRNLEV.GE.1) WRITE(OUTU,*), 'VXI 4 = ', VX(I)
ENDDO
```

```fortran
IF(IGAUSS.GT.0) GOTO 101
  V=SD
  Y = RANDOM(IG)
  IF(Y.LE.0.5) V=-SD
  RETURN
101 CONTINUE
  A=SQR2(MINTWO+LOG(RANDOM(IG))
  B=TWOP*PI*RANDOM(IG)
  V=SD*A+COS(B)*AM
```
Programming issues

- VXI is updated in 24 places across 7 files
  - Find the operations involved in the updating of VXI and determine the range of values in those operations

Hypothesis 2 is likely to be the cause of discrepancies, but the testing needs to be extended in order to track different cases
Hypothesis (3)

- Architecture dependent issues. Processors of the IA32 architecture (Intel 386, 486, Pentium and compatibles) feature a floating-point unit often known as x87 [Int 2005]
  - x87 has 80 bit registers in double extended format
  - The most usual way of generating code for the IA32 is to hold temporaries
  - It is not always possible to do everything inside registers, and compilers generally spill extra temporary values to main memory

Potential problems with x87

Spilling of registers

- For example a double temporary (80 bits) will be spilt to a 64-bit double precision memory cell.
  - Then the final result of the computations depend on how the compiler allocates registers, since temporaries may or may not be rounded, depending if they are spilt to main memory.
Double rounding (1)

- For example: results computed in the long double 80-bit registers of the x87, then rounded to the IEEE double precision type for storage in memory.
  - In round-to-0, round-to+$\infty$ and round-to-$-\infty$ modes, this is not a problem
  - In round-to-nearest mode, there exist some borderline cases where differences can arise

\[
y = \text{op}(x) \in \mathbb{R}
\]

\[
y = (1|0)^{63}1
\]

\[
y = (1|0)^{63}0
\]
Double rounding (2)

- Double rounding can also cause some differences for very small numbers that are rounded into subnormal double-precision values if computed in IEEE-754 double precision
  - These numbers will be rounded to normalized values inside the FPU register, because of a wider range of negative exponents
  - Then they will be rounded again into double-precision subnormals when written to memory
- This is known as double-rounding on underflow.
More experiments are needed and reproducibility of the discrepancies found at docking@home are required to test this hypothesis.

Hypothesis 3 is an extension of hypothesis 1. Hypothesis 2 and 3 are complementary, and are the most likely causes of discrepancies in charmmb.
Possible directions

- Testing of hypotheses 2 and 3 using the docking@home infrastructure
- Identify all the possible points of discrepancies and the functions involved
  - If it is possible to get a reduced set of functions and rewrite them in terms that minimize discrepancies
  - Otherwise keep playing with the compiler to get a charmm version that minimizes the occurrence of discrepancies
W Kahan’s web page: http://www.cs.berkeley.edu/~wkahan/

Floating point issues:
http://www.linuxtopia.org/online_books/an_introduction_to_gcc/gcc_intro_70.html

What Every Computer Scientist Should Know About Floating-Point Arithmetic
http://docs.sun.com/source/806-3568/ncg_goldberg.html

Tutorial to Understand IEEE Floating-Point Errors
http://support.microsoft.com/kb/42980/

Floating-Point Computing: A Comedy of Errors?
http://developers.sun.com/solaris/articles/fp_errors.html

Differences Among IEEE 754 Implementations
http://www.validlab.com/goldberg/addendum.html

IEEE Arithmetic
http://docs.sun.com/source/806-3568/ncg_math.html

Avoiding floating point errors
http://www.fortran.gantep.edu.tr/90/unfp/ch4-9.html
References

- David Monniaux, *The pitfalls of verifying floating-point computations*, *ACM Transactions on Programming Languages and Systems* 30, 3 (2008) 12