LINUX on System z:
Explore Decimal Floating Point support in Linux on System z

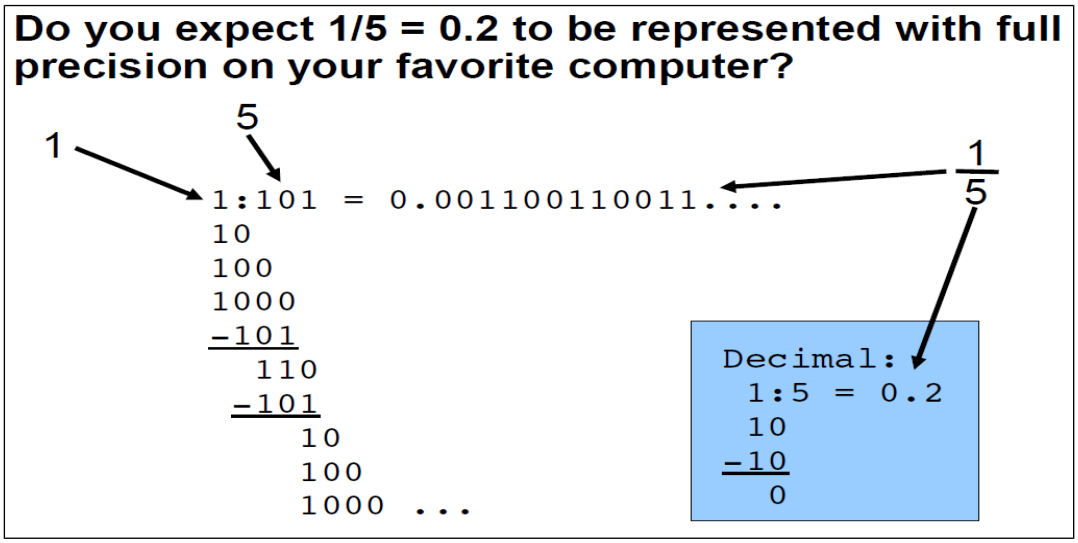
Version 2.2

February 27, 2014

Mario Held, IBM Research & Development, Mario.Held at de.ibm.com
Andreas Krebbel, IBM Research & Development, Andreas.Krebbel at de.ibm.com

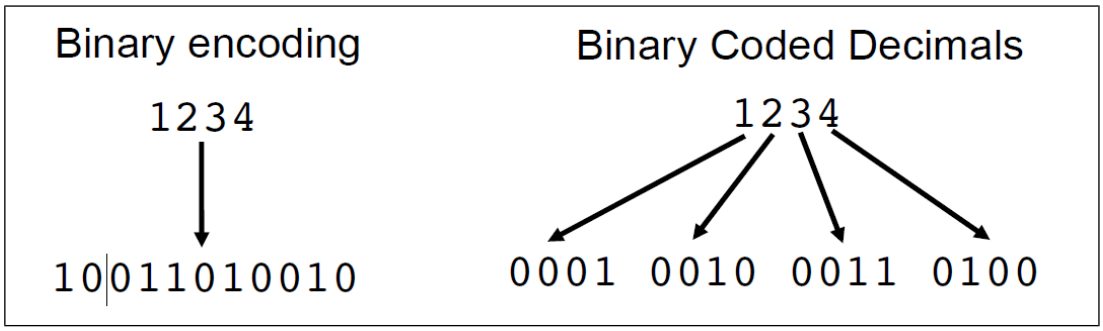
Motivation to use Decimal Floating Point (DFP)

We have had integer and floating point data types in computer languages like C, C++ and FORTRAN to store numerical values. Unfortunately binary floating point has issues to store the values with the accuracy the user expects. Even worse, when making many calculations the inaccuracy builds up.



More examples are shown here http://speleotrove.com/decimal/decifaq1.html#inexact

So for a long time, financial, commercial and user-centric applications and web services have been using binary coded decimal (BCD) format. Beside the big advantage of keeping accuracy in financial calculations, BCD also has some disadvantages. The representation of BCD values is similar to strings and every digit is encoded separately. Storing and operations on data consume slightly more resources. Only fix point arithmetic is supported and the operands have to reside in memory, not in registers.



Decimal floating point (DFP) was introduced to improve the situation and to combine the advantages of binary floating point and binary coded decimal. The term DFP covers a decimal (base 10) encoding of numeric values as well as operations on the numbers.

Advantages and Disadvantages of DFP:

- **Offers the required accuracy for financial, commercial and user-centric applications**
- Uses Densely Packed Decimal encoding to save memory.
- Hardware support for DFP has been available with the System z9-EC and newer machines. DFP operations run much faster when supported by hardware compared to the software solution.
- Exploits a new industry standard for calculations (754-2008 IEEE Standard for Floating-Point Arithmetic) which makes it easier to communicate and interchange data between various programs and software products hosted on various computer systems.
- Calculations using DFP format and operations need slightly more resources than using binary floating data (memory footprint, complexity of operations).

Prerequisites:

Software support requirements (mandatory):

SLES11 SP1 was the first available SUSE Linux Enterprise Server distribution which includes the needed softwares packages at the required level. More mature dfp support was included in SLES11 SP2. RHEL 6 was the first Red Hat distribution which includes DFP support on the mainframe. The latest public available updates are RHEL6.1 and RHEL6.2

- **GCC** – system GNU C Compiler with integrated DFP support

SUSE gcc-4.3-62.198 (SLES11 SP2) and higher

RHEL gcc-4.4.6-3.el6.s390x (RHEL6.2) and higher

- **libdfp** - library containing support routines for calculating and printing decimal floating point values. Provides hardware-accelerated overrides of software DFP functions. A hardware and a software version will be installed in order to choose dynamically. libdfp is maintained by IBM as external library. Includes limited software math library support.

SUSE libdfp1-1.0.7-0.7.9 (SLES11 SP2) and higher

RHEL libdfp-1.0.6-2.el6.s390x (RHEL6.2) and higher

- **glibc** - GNU C Library provides the most important standard libraries used by nearly all programs: the standard C library, the standard math library, and the POSIX thread library. A recent glibc is needed to implement printf support externally.

SUSE glibc-2.11.3-17.31.1 (SLES11 SP2) and higher

RHEL glibc-2.12-1.47.el6.s390x (RHEL6.2) and higher

- **binutils** - C compiler utilities: ar, as, gprof, ld, nm, objcopy, objdump, ranlib, size, string and strip. A recent bintutils version is needed to support the DFP instruction set.

SUSE binutils-2.21.1-0.7.25 (SLES11 SP2) and higher

RHEL binutils-2.20.51.0.2-5.28.el6.s390x (RHEL6.2) and higher

- **gdb** – the GNU debugger to debug applications including DFP instructions

SUSE gdb-7.3-0.8.4 (SLES11 SP2) and higher

RHEL gdb-7.2-50.el6.s390x (RHEL6.2) and higher

Hardware support requirements (recommended for best performance):

Hardware support is available with IBM System z9-EC (System z9 GA3), IBM System z10, IBM zEnterprise 196 (z196) and IBM zEnterprise EC12 (zEC12).

Quick check for hardware support on System z (example on zEC12, looks similar on z196, z10 and z9-ec). Look for 'dfp' in the features line:

```
[root@p10lp06 ~]# cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 1
bogomips per cpu: 18115.00
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te
cache0          : level=1 type=Data scope=Private size=96K line_size=256 associativity=6
cache1          : level=1 type=Instruction scope=Private size=64K line_size=256 associativity=4
cache2          : level=2 type=Data scope=Private size=1024K line_size=256 associativity=8
cache3          : level=2 type=Instruction scope=Private size=1024K line_size=256 associativity=8
cache4          : level=3 type=Unified scope=Shared size=49152K line_size=256 associativity=12
cache5          : level=4 type=Unified scope=Shared size=393216K line_size=256 associativity=24
processor 0: version = 00,  identification = 06AA24,  machine = 2827
[root@p10lp06 ~]#
```

Usage:

Language C:

- The C standard includes the following new data types (these are native data types just like integer, long and float, double etc): _Decimal32,_Decimal64,_Decimal128
- New constant suffixes: df, dd, dl, DF, DD and DL
- Arithmetic operations (add, subtract, multiply, divide, sqrt, … sinus...)
- String to DFP conversions: strtod32, strtod64, strtod128
- Set and get rounding mode functions (`fe_dec_setround` and `fe_dec_getround`)
- New printf format letters: %Hf, %Df, %Ddf
- Function scanf is not yet supported

Build the code:

- **Fastest:** gcc -O3 **-march=<z9-ec | z10 | z196 | zEC12> -ldfp …** Building a binary exploiting hardware DFP application which is only able to run on the specified target machine or newer machine. Usage of new instructions and special scheduling for the target machine is used. Which values are supported for parameter -march depends on the version

of the used GCC compiler in your Linux distribution. It is not necessary to specify **-mhard-dfp** because it is the default.

- **Preferred for compatibility:** Building a DFP application which should run also on machines prior to z9 GA3 (without hardware DFP support): **gcc -O3 -ldfp …** On a machine supporting hardware DFP, the software functions will dynamically be replaced by hardware versions.
- **Enforce software dfp:** gcc -O3 **-mno-hard-dfp …** The -mno-hard-dfp switch can be used to to make GCC fall back to software DFP support even with -march=z9-ec or higher. This combination of options seems only relevant in test scenarios.

Usage with Java:

The java.math.BigDecimal class provides software DFP support. Decimal Floating Point data types are supported. The Java support has been available since Java 1.5. Because of performance improvements and lots of fixes in releases and service packs, the use of the latest Java package usable with your Linux distribution is recommended.

Test methods for DFP:

There are several sources to find both accuracy and performance tests for DFP in the Internet.

Accuracy Test:

Various test suites are available. For example

General Decimal Arithmetic test-cases Version 2.44 – 7 Apr 2009

and

Floating-Point Test Generator – Fpgen can be downloaded from the Internet.

Performance Test in C:

Tests were completed using the cTelco benchmark. This test was carried out on Linux on IBM System zEC12. The benchmark is available here The ‘telco’ benchmark.

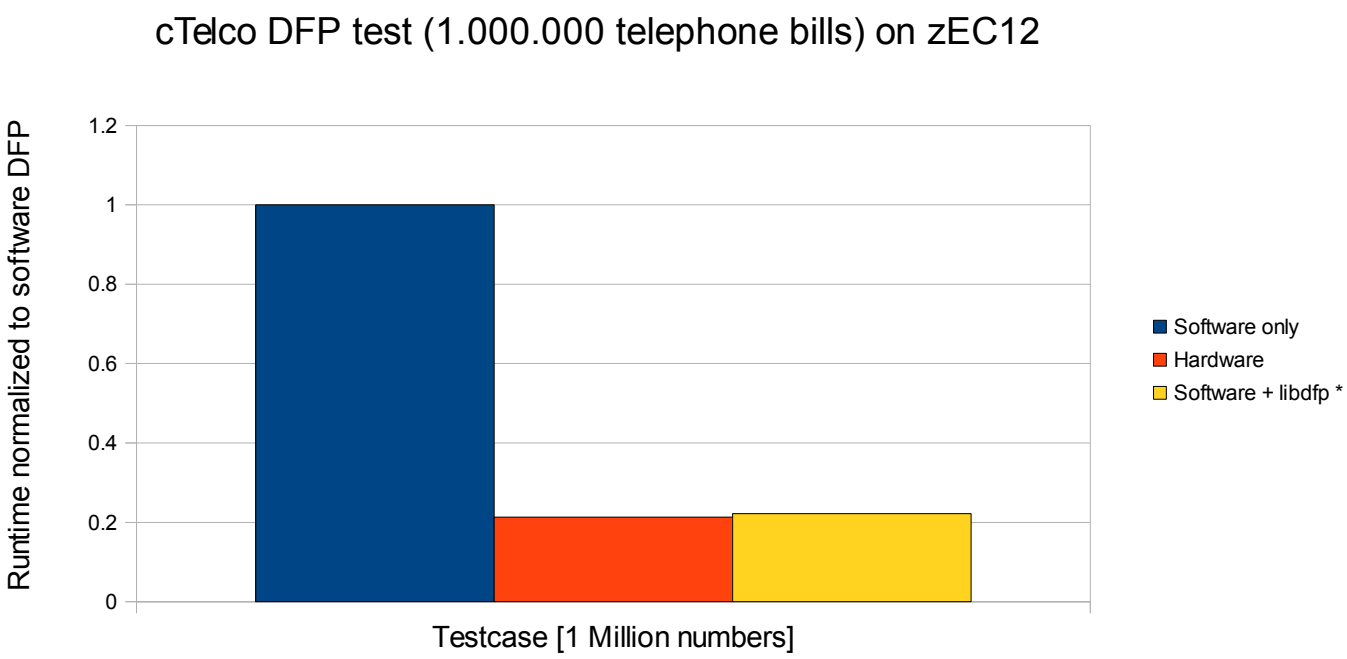
The cTelco benchmark models a telephone company's billing system. The workload comprises billing of one million telephone calls including tax using DFP arithmetics. The test giving us the following results was carried out using SUSE SLES11 SP3 running on a zEC12 (gcc version gcc43-4.3.4_20091019-0.37.30, glibc-2.11.3-17.54.1 and libdfp1-1.0.8-0.7.28 and libdfp-devel-1.0.8-0.7.28).

The benchmark code was compiled in three different ways:

DFP software solution:	gcc -O3 -march=z10 -mno-hard-dfp -o cTelco_gcc cTelco754r_gcc.c
DFP hardware support enabled (default):	gcc -O3 -march=z10 -o cTelco_gcc cTelco754r_gcc.c
DFP hardware support only via libdfp:	gcc -O3 -march=z10 -mno-hard-dfp -ldfp -o cTelco_gcc cTelco754r_gcc.c

If DFP hardware support is exploited, the performance improves significantly. We see that this benchmark completes in approximately a fifth of the time when accessing hardware support directly or via a hardware-accelerated library.

The advantage of the hardware DFP exploitation (direct or via hardware accelerated libdfp) can be seen compared with the software solution running on the same system in the following chart. Lower bars show the better results.



* libdfp provides also hardware support

Performance Test in Java:

The Java variant of the Telco benchmark jTelco was tested on SLES11 SP3 using the Java package java-1_7_0-ibm-1.7.0_sr4.1-0.5.1 as included Java(TM) SE Runtime Environment (build pxz6470sr4fp1-20130325_01(SR4 FP1)) IBM J9 VM (build 2.6, JRE 1.7.0 Linux s390x-64) on IBM System zEC12..

```
p10lp06: # java jTelco expon180.1e6b
telco Java benchmark; processing 'expon180.1e6b'
-- telco Java benchmark result --
   1000000 numbers read from 'expon180.1e6b'
   Time per number       0.656us  [1000000]
--
   sumT = 1004737.58
   sumB = 57628.30
   sumD = 25042.17
```

The results show a higher standard deviation when running several times.

Further reading and DFP resources:

754-2008 IEEE Standard for Floating-Point Arithmetic

General Decimal Arithmetic by Mike Cowlishaw

C language extension TR24732 ratified extension

C++ language extension TR 24733 ratified extension

IBM Research & Development GmbH
Schoenaicher Str. 220
71032 Boeblingen
Germany
Produced in Germany,
02/2014

All statements regarding IBM’s future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Performance is based on measurements and projections using benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance results stated here.