Strict Suboptions

Ian McIntosh
Toronto IBM Lab

Agenda

- Why Strict Suboptions Are Needed
- Overall Suboptions Design
- General Suboptions
- Detail Suboptions
- Examples
- Pragmas and Directives
- Interacting and Related Options
- Options Listing
- Q&A

Why Strict Suboptions Are Needed

- **Using `nostrict` allows many programs to run faster.**
- **Using `strict` is sometimes necessary for correctness.**
- **Some programs would run faster with `nostrict` for some compiler decisions but need `strict` for others.**
- **`nostrict`-related problem investigation would be easier with more detailed controls.**

Overall Suboptions Design

- **Detail Level:**
  - **Some users want general high level controls,**
  - **other users need very specific low level controls,**
  - **so different views of the same controls are needed.**
  - **Hundreds of decisions are controlled by `strict`/`nostrict`,**
  - **so most suboptions will control groups of decisions.**
  - **Future versions may need even more detailed suboptions.**
- **Constraints:**
  - **`strict` suboptions control the same things `strict` did.**
  - **Future versions may control more decisions,**
  - **so may need more suboptions.**
- **As simple as will meet users' needs.**

Overall Suboptions Design

- `-qstrict` -- same as before
- `-qnostrict` -- same as before

- `-qstrict=all` -- same as `-qstrict`
- `-qstrict=none` -- same as `–qnostrict`

- `-qstrict=`*X* -- be strict about *X* (*X* is any `strict` suboption)
- `-qstrict=no`*X* -- be relaxed about *X*

- `-qstrict=`*X*`:`*Y*`:`*Z* -- be strict or relaxed about *X*, *Y* and *Z*
  (*X* and/or *Y* and/or *Z* may have “no” prefix)

- `-qstrict=`*X* `\`
  `-qstrict=`*Y*`:`*Z* -- be strict or relaxed about *X*, *Y* and *Z*
  (*X* and/or *Y* and/or *Z* may have “no” prefix)

General Suboptions

- `-qstrict=all` -- be strict or
- `-qstrict=none` -- relaxed about all changes

- `-qstrict=precision` -- be strict or
- `-qstrict=noprecision` -- relaxed about changing precision

- `-qstrict=exceptions` -- be strict or
- `-qstrict=noexceptions` -- relaxed about changing exceptions (whether more or less or moved)

`[no]exceptions` does not control *everything* that could produce different results, so different exceptions are possible even with `-qstrict=noexceptions`.

- **Each general suboption controls decisions itself, and also controls nested suboptions.**

General Suboptions

- `-qstrict=ieeefp` -- be strict or
- `-qstrict=noieeefp` -- relaxed about violating IEEE 754

  `[no]ieeefp` controls individual operations defined by IEEE 754, not how operations interact or are ordered.

  Detail suboptions allow controlling specific aspects of `[no]ieeefp`. Most operations are affected by multiple detail suboptions, and also by `[no]exceptions`.

- `-qstrict=order` -- be strict or
- `-qstrict=noorder` -- relaxed about operation order

  `[no]order` controls the order between operations, as defined by language semantics, not how each operation is implemented.

  Detail suboptions allow controlling specific aspects of `[no]order`.

Detail Suboptions – ieeefp group

- `-qstrict=nans` -- be strict or
- `-qstrict=nonans` -- relaxed about calculations with NaNs
  eg, `x+0.` => `x` is wrong if `x` is a NaN

- `-qstrict=infinities` -- be strict or
- `-qstrict=noinfinities` -- relaxed about calculations with infinities
  eg, `x*0.` => `0.` is wrong if `x` is an infinity

- `-qstrict=subnormals` -- be strict or
- `-qstrict=nosubnormals` -- relaxed about calculations with subnormals
  eg, `x/10.` => `x*.1` overflows if `x` is subnormal

- `-qstrict=zerosigns` -- be strict or
- `-qstrict=nozerosigns` -- relaxed about calculations with sign of zero
  eg, `(-a)-b*c` => `-(a+b*c)` may give the wrong zero

- `-qstrict=operationprecision` -- be strict or
- `-qstrict=nooperationprecision` -- relaxed about operation precision
  eg, `x/5.` => `x*.2` is not precise

Most operations need more than one of these; some also need `noexceptions`.

Detail Suboptions – order group

- `-qstrict=association` -- be strict or
- `-qstrict=noassociation` -- relaxed about changing association order

  eg, `a+b+c`, `(a+b)+c`, `a+(b+c)`, `(a+c)+b`
  may give different results

- `-qstrict=reductionorder` -- be strict or
- `-qstrict=noreductionorder` -- relaxed about reductions like dot product

  changing the order of evaluation
  (and hence the order of rounding)

- `-qstrict=guards` -- be strict or
- `-qstrict=noguards` -- relaxed about moving variable use

  past `if` / `for` / `while` / `call`

  eg, `if(k>=0) x=y[k]` => `t=y[k]; if(k>=0) x=t`

  is faster but risky if `k` is not a valid subscript

Detail Suboptions – miscellaneous

- `-qstrict=library` -- be strict or
- `-qstrict=nolibrary` -- relaxed about replacing library functions
  eg, libm `sin(x)` => MASS `sin(x)`
  may give different results

- `-qstrict=constructcopy` -- be strict or
- `-qstrict=noconstructcopy` -- relaxed about constructing
  Fortran REAL arrays
  via a temporary array
  to ensure atomic construction or not
  in case of exceptions

Suboption Grouping

- **`all` and none include:**
  everything.
- **`precision` includes:**
  `subnormals`, `operationprecision`, `association`, `reductionorder`, `library`,
  and controls some things itself.
- **`exceptions` includes:**
  `nans`, `infinities`, `subnormals`, `guards`, `library`, `constructcopy`,
  and controls some things itself.
- **`ieeefp` includes:**
  `nans`, `infinities`, `subnormals`, `zerosigns`, `operationprecision`.
- **`order` includes:**
  `association`, `reductionorder`, `guards`.

Basic Suboption Grouping

all none	ieeefp	nans
		infinities
		subnormals
		zerosigns
		operationprecision
	order	association
		reductionorder
		guards
	library	
	constructcopy (Fortran only)	

Precision Suboption Grouping

all none	precision	subnormals
		operationprecision
		association
		reductionorder
		library

Exceptions Suboption Grouping

all none	exceptions	nans
		infinities
		subnormals
		guards
		library
		constructcopy (Fortran only)

Examples

- `-qstrict=all:nooperationprecision:noreductionorder`

- **Be strict about everything,**

- **except relaxed about `operationprecision`,**
  part of what's needed to allow
  `x / loop_constant => x * (1 / loop_constant),`
  which is needed to allow faster `MOD`
  (note this allows other changes too),

- **and relaxed about `reductionorder`,**
  to allow recognizing dot product and similar reductions
  and generating faster parallelized code, without allowing other reordering.

Examples

```
do i = 1, n
    a(i) = b(i) / x
end do
```

`-qstrict=operationprecision\`
`:exceptions:zerosigns`

```
  LFL     fp0=x(gr31,0)

    . . .
CL.30:
  LFL     fp2=b[](gr4,8)
  AI      gr3=gr3,8
  STFL    a[](gr3,0)=fp1
  DFL     fp1=fp2,fp0,fcr   =b(i)/x
  AI      gr4=gr4,8
  BCT     ctr=CL.30
```

divide by x

`-qstrict=nooperationprecision\`
`:noexceptions:nozerosigns`

```
  LFS     fp0=+CONSTANT_AREA(gr5,4) =1
    . . .
  LFL     fp2=x(gr31,0)
  RCPFL   fp0=fp0,fp2,fcr           =1/x
    . . .
CL.30:
  AI      gr3=gr3,8
  LFL     fp2=b[](gr4,8)
  AI      gr4=gr4,8
  STFL    a[](gr3,0)=fp1
  MFL     fp1=fp2,fp0,fcr   =b(i)*(1/x)
  BCT     ctr=CL.30
```

multiply by reciprocal of x

Examples

```
dp = 0.
do i = 1, n
    dp = dp + a(i) * b(i)
end do
```

Examples

-qstrict=reductionorder

```
CL.39:
  LFS    fp2=b(gr4,36)
  LFS    fp1=a(gr3,4)
  AI     gr4=gr4,32
  FMAS   fp0=fp0,fp1,fp2,fcr
  LFS    fp2=b(gr4,8)
  LFS    fp1=a(gr3,8)
  FMAS   fp0=fp0,fp1,fp2,fcr
  LFS    fp1=a(gr3,12)
  LFS    fp2=b(gr4,12)
  FMAS   fp0=fp0,fp1,fp2,fcr
  LFS    fp1=a(gr3,16)
  LFS    fp2=b(gr4,16)
  FMAS   fp0=fp0,fp1,fp2,fcr
  AI     gr3=gr3,32
  LFS    fp2=b(gr4,20)
  LFS    fp1=a(gr3,-12)
  FMAS   fp0=fp0,fp1,fp2,fcr
  LFS    fp1=a(gr3,-8)
  LFS    fp2=b(gr4,24)
  FMAS   fp0=fp0,fp1,fp2,fcr
  LFS    fp2=b(gr4,28)
  LFS    fp1=a(gr3,-4)
  FMAS   fp0=fp0,fp1,fp2,fcr
  LFS    fp1=a(gr3,0)
  LFS    fp2=b(gr4,32)
  FMAS   fp0=fp0,fp1,fp2,fcr
  BCT    ctr=CL.39
```

Single serial accumulator – slow but exact

-qstrict=noreductionorder

```
CL.40:
  LFS    fp6=b(gr6,16)
  LFS    fp10=b(gr6,36)
  FMAS   fp29=fp29,fp23,fp26,fcr
  LFS    fp27=b(gr6,20)
  FMAS   fp30=fp30,fp22,fp21,fcr
  FMAS   fp31=fp31,fp1,fp0,fcr
  LFS    fp25=a(gr5,-8)
  LFS    fp24=b(gr6,24)
  LFS    fp23=a(gr5,-4)
  LFS    fp21=b(gr6,32)
  LFS    fp26=b(gr6,28)
  AI     gr6=gr6,32
  LFS    fp1=a(gr5,4)
  FMAS   fp9=fp9,fp3,fp8,fcr
  LFS    fp22=a(gr5,0)
  FMAS   fp12=fp12,fp2,fp7,fcr
  FMAS   fp11=fp11,fp4,fp6,fcr
  LFS    fp3=a(gr5,8)
  LRFL   fp0=fp10
  LFS    fp8=b(gr6,8)
  FMAS   fp13=fp13,fp5,fp27,fcr
  LFS    fp2=a(gr5,12)
  LFS    fp7=b(gr6,12)
  LFS    fp5=a(gr5,20)
  LFS    fp4=a(gr5,16)
  AI     gr5=gr5,32
  FMAS   fp28=fp28,fp25,fp24,fcr
  BCT    ctr=CL.40
    . . .
```

8 parallel accumulators – fast but inexact

Pragmas and Directives

- **XL C/C++ #pragma option_override**

```
#pragma option_override (fname,"opt(strict,all,noreductionorder)")
```

- **XLF @PROCESS directive**

```
@PROCESS STRICT(ALL, NOREDUCTIONORDER)
```

These allow:

- **Putting required suboptions in the source file.**
- **Having suboptions control specific functions or subroutines instead of everything in all the files compiled together.**

Interacting Options

- `-qstrict` sets `-qfloat=nofltint`
- `-qnostrict` sets `-qfloat=fltint`
  so do `-qstrict=[no]operationprecision` or/and `[no]exceptions`

- `-qstrict` sets `-qfloat=norsqrt`
- `-qnostrict` sets `-qfloat=rsqrt`
  so do `-qstrict=[no]infinities` or/and `[no]operationprecision` or/and `[no]exceptions`

- `-qstrict` sets `-qfloat=rngchk`
  so do `-qstrict=[no]nans` or `[no]infinities` or `[no]zerosigns` or `[no]exceptions`

- `-qfloat=relax`
  affects some of the same things as `-qstrict=noieeefp`

Related Options

- **Other options which `strict/nostrict` users may have some interest in:**

`-qsmp`	
`-qfloat=maf`	`-qessl`
`-qfloat=strictnmaf`	`-qhot=vector`
`-qfloat=rndsngl`	`-qhot=simd`
`-qfloat=hsflt`	`-qenablevmx`
`-qfloat=hscmplx` (xlf)	`-qalias=...`
`-qfloat=hssngl`	`-qassert=...` (xlf)
`-qfloat=single` (xlf)	`-qxlf90=signedzero` (xlf)
`-qfloat=fltint`	`-qxlf2003=signdzerointr` (xlf)
`-qfloat=nans`	`-qstrictieeemod` (xlf)
`-qfloat=spnans`	`-qport=sce` (xlf)
`-qfloat=rrm`	`-qswapomp` (xlf)
`-qstrict_induction`	`-qfloat=fenv` (xlf)

Options Listing

- `-qstrict` => `STRICT` (as before)
- `-qnostrict` => `NOSTRICT` (as before)

- `-qstrict=all` => `STRICT=ALL`
- `-qstrict=none` => `STRICT=NONE`

- `-qstrict=`*X*`:`*Y* => `STRICT=[ALL|NONE]:`*X*`:`*Y*`:all “no” suboptions`

- `-qlistopt` => `STRICT=`all strict suboptions

- Fortran listing syntax is `STRICT(X,Y,Z)` not `STRICT=X:Y:Z`.

- With `-qsaveopt`, options are also put in the object file.

Questions and Answers