UCLouvain
Institut de recherche en mathématique et physique
Centre de Cosmologie, Physique des Particules et Phénoménologie

Fortran versus C++ speed

Olivier Mattelaer

MCnet

Motivation

- MG5aMC is the **only code** of MCnet still relying in Fortran
- **Interface**
  - ➡ lhapdf is currently a nightmare.
  - ➡ Can be easier to link to Parton-Shower
- This small comparison was originally trigger by the question about using CudaC or CudaFortan
  - ➡ No real plan to move to C

- Results can be interesting for other MCnet tools
  - ➡ This is the reason for this lightning talk

Standalone mode

- MG5aMC can create simple code that ONLY evaluates the amplitude square.
  - ➡ Linked to Rambo for the PS generation
  - ➡ Code available in c++ and in fortran
    - Code not 100% identical
      - Fortran code has a layer of optimisation to reduce RAM usage (was found irrelevant for the speed)
  - ➡ The comparison therefore is directly linked to the speed of evaluation of the matrix-element
    - And therefore to the speed of double precision **complex number arithmetic**
    - Not same as a Parton-shower arithmetic

Setup

- Process: g g > t t~ g g
  - ➡ 10 thousands phase-space evaluated
  - ➡ Timing include rambo timing
- Timing presented here with a quite old gcc version (4.8)
  - ➡ Findings confirmed with more recent version of gcc/ different machines.

Result

Compilation flag	CPP	F77
-O	27 s	9.48 s

- C++ is **three times slower**

Result

Compilation flag	CPP	F77
-O	27 s	9.48 s

- C++ is **three times slower**
  - ➡ **PY8** author is aware of that:
    - Stefan gives mod a user class implementing complex number

Timing for code/flag

	CPP	Complex hack	F77
-O	27	9.1	9.48

- C++ is **three times slower**
  - ➡ **PY8** author is aware of that:
    - Stefan gives mod a user class implementing complex number
    - Fix indeed the issue
      - **How is this possible?**

Timing for code/flag

	CPP	Complex hack	F77
-O	27	9.1	9.48
-O3	26.04	8.9	9.3

- O3 is full optimisation
  - ➡ Including hardware specific
    - ➡ **Small gain if at all**

Timing for code/flag

	CPP	Complex hack	F77
-O	27	9.1	9.48
-O3	26.04	8.9	9.3
-Ofast	8.0	8.6	6.4

- Ofast
  - ➡ optimizations that are not valid for all standard-compliant programs
  - ➡ Pythia hack not needed anymore
    - ➡ Fortran still faster but reasonable

Timing for code/flag

	CPP	Complex hack	F77
-O	27	9.1	9.48
-O3	26.04	8.9	9.3
-Ofast	8.0	8.6	6.4

- C++ is more careful than fortran in the handling of nan
  - ➡ GCC has dedicated linked to that
    - Fcx-fortan-rules
    - Fcx-limited-range

Timing for code/flag

	CPP	Complex hack	F77
-O	27	9.1	9.48
-O3	26.04	8.9	9.3
-Ofast	8.0	8.6	6.4
-0 -fcx-fortran-rules -fcx-limited-range	9.27	9.4	9.3

Conclusion

Winner

- Fortran is faster
  - ➡ without any compilation flag
  - ➡ With most agressive flag

Message

- Compiler flag are important
- Using Black Box can hurt you

Remark

1) On Mac clang does not support the gcc flag for speeding up complex number

`-fcx-limited-range`

When enabled, this option states that a range reduction step is not needed when performing complex division. Also, there is no checking whether the result of a complex multiplication or division is `NaN + I*NaN`, with an attempt to rescue the situation in that case. The default is `-fno-cx-limited-range`, but is enabled by `-ffast-math`.

This option controls the default setting of the ISO C99 `CX_LIMITED_RANGE` pragma. Nevertheless, the option applies to all languages.

`-fcx-fortran-rules`

Complex multiplication and division follow Fortran rules. Range reduction is done as part of complex division, but there is no checking whether the result of a complex multiplication or division is `NaN + I*NaN`, with an attempt to rescue the situation in that case.

The default is `-fno-cx-fortran-rules`.