99-coding-dojo

June 2, 2024

1 Coding Dojo

Starting from the program below, each participant has to take turns proposing an improvement/modification that uses certain elements of the remainders and/or the new syntaxes.

Some ideas : - replace some conversion with a `static_cast`, - use east const, - replace old-fashioned loops with range-based for, - use universal initialisation with `{}`, - infer some types with `auto`, `decltype`, etc… - replace hand-made dynamic memory management with smart pointers, - add `explicit` to some constructor, - template the code so to try float and double precision in the same program, - check some static properties with `static_assert` and type traits, - try to precompute things at compile time with `constexpr`, - try to save some useless copies with rvalue references `&&`, - use lambda functions together with algorithmes from the standard library, - insert some variadic templates…

[13]:
```
%%file tmp.dojo.cpp

#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>

// complex numbers
template< typename R >
class Cplx {
  public :
    Cplx( R a_real = 0., R a_imag = 0. ) : m_real(a_real), m_imag(a_imag) {
    }
    friend Cplx<R> operator*( Cplx<R> const & lhs, Cplx<R> const & rhs ) {
      R r = lhs.m_real*rhs.m_real - lhs.m_imag*rhs.m_imag ;
      R i = rhs.m_real*lhs.m_imag + lhs.m_real*rhs.m_imag ;
      return Cplx<R>(r,i) ;
    }
    friend std::ostream & operator<<( std::ostream & os, Cplx<R> const & c ) {
      return (os<<std::setprecision(2)<<c.m_real<<std::showpos<<c.m_imag<<"j") ;
    }
  private :
    R m_real, m_imag ;
 } ;
```

```
// alias
typedef Cplx<double> CplxD ;

// return a random complex on the unit circle
CplxD random_complex() {
  double e = 2*M_PI*(double(std::rand())/RAND_MAX) ;
  return CplxD(std::cos(e),std::sin(e)) ;
}

// fill an array of random complexes
void init( CplxD * cs, std::size_t dim ) {
  std::srand(1) ;
  for ( std::size_t i = 0 ; i < dim ; ++i ) {
    cs[i] = random_complex() ;
  }
}

// return a single complex ^ degree
CplxD power( CplxD c, int degree )
 {
  CplxD res(1.,0.) ;
  for ( int d = 0 ; d < degree ; ++d ) {
    res = res*c ;
  }
  return res ;
 }

// raise an array of Cplxs to power degree
void process( CplxD * cs, std::size_t dim, int degree ) {
  for ( std::size_t i = 0 ; i < dim ; ++i ) {
    cs[i] = power(cs[i],degree) ;
  }
}

// return the global product of n complex numbers
CplxD reduce( const CplxD * cs, std::size_t dim ) {
  CplxD res(1.,0.) ;
  for ( std::size_t i = 0 ; i < dim ; ++i ) {
    res = res*cs[i] ;
  }
  return res ;
}

// main program
int main() {
```

```
  // general parameters
  const std::size_t DIM = 1000000 ;
  const int DEGREE = 16 ;

  // allocate memory
  CplxD * cs = new CplxD [DIM] ;

  // generate random input
  init(cs,DIM) ;

  // compute output
  process(cs,DIM,DEGREE) ;

  // check result
  std::cout<<reduce(cs,DIM)<<std::endl ;

  // release memory
  delete [] cs ;

  // end
  return 0 ;
}
```

```
Overwriting tmp.dojo.cpp
```

[22]:
```
!rm -f tmp.dojo.exe && g++ -std=c++03 -O2 tmp.dojo.cpp -o tmp.dojo.exe
```

[23]:
```
!\time -f "%U s" ./tmp.dojo.exe
```

```
0.068-1j
0.15 s
```