Modern C++ approaches to FEM modelling

A. M. Gurin[1]

[1]*Lavrentyev Institute of Hydrodynamics SB RAS, Novosibirsk, Russia*

State of the art R&D demands complex, coupled simulations of the physical phenomena. Modern research goals could require numerical simulation with unstructured meshes, dynamic topology, nonlinearity, high order elements, different equations in subdomains. This complex tasks provide considerable challenge for programmers. Old ways of programming do not provide adequate solution for this complex tasks anymore. To deal with the mentioned problems one has to choose an appropriate tool. At present day C++ is one of the most popular programming languages for scientific computations. It has a wide variety of libraries for common tasks such as solution of linear systems and formidable performance rivaled only by Fortran. In last dozen of years C++ underwent quite extensive modification which made it a performant yet flexible, multiparadigm language for scientific computation. Programming techniques and features such as lambda functions, functional programming and template metaprogramming have become available or much easier to use in C++11/C++14/C++17.

Newest features of C++ were assessed and tested on simple computational problems. Possible performance impact and optimization features were investigated by studying of the assembly code generated by C++ compilers gcc, clang, MSVC. Compiler inlining capabilities were proven on a simple problem of solving an equation using the Newton method. Modern compilers aggressively optimize the code and are capable of actually solving the equation already in the compilation process. The compiler will also mathematically simplify all parts of the algorithm wherever possible if there is not enough information given to solve the equation. To illustrate advantages of this technique and to show that it is applicable to complex real world problems, the elasticity problem has been solved using the functional approach and with the help of template metaprogramming.

Improved static polymorphism and template metaprogramming capabilities of C++ are powerful tools that let us write flexible generic code with better code reuse. This tools are applied on compile time so there is no negative impact on the runtime performance. Quite the contrary, performance is further improved by the additional data that the compiler can use to optimize the program. Code that has been written in this manner is less prone to errors because most of them are discovered during the compilation phase.