The Adaptiv Framework

Nuno Alves de Sousa

Instituto Superior Técnico
Área Científica de Mecânica Aplicada e Aeroespacial

February 17, 2019

Outline

Outline

Attributes of good software

"Software and cathedrals are much the same – first we build them, then we pray."

(Anonymous)

Attributes of good software

Not *what* the program does, but *how well* it does it:

Maintainability	reduce/reverse "code entropy" cheaper/safer to change than to rewrite
Dependability	availability, reliability, safety, integrity
Efficiency	algorithmic efficiency storage efficiency
Usability	"consumer" effectiveness and efficiency elegance and clarity perceived by the user

Outline

Where to start?

"What happens before one gets to the coding stage is often of crucial importance to the success of the project." (Meek & Heath - Guide to Good Programming Practice)

Higher-level prerequisites to provide a solid foundation for coding:

- Coding standards
- Choice of programming language
- Life cycle, architecture, design
- Requirements

Coding standards

Coding conventions are particularly important in collaborative projects:

- Much easier to read someone else's code
- Uniform style (*e.g.* naming conventions for filenames, variables, etc)
- Deal with undereducated programmers
- Avoid insufficient library use
- Portability
- Commenting conventions:
  - Speed up knowledge transfer
  - Comment only what code expresses poorly (intent)
  - Comments lie, code never lies
  - Do not comment code modifications (use a version control system)

Version control

Source code is the most valuable asset of any software project

Version control systems (VCS)

- Management of changes to all non-binary files
- Complete retrace of all versions of each file
- History of the authors of such changes

Critical advantages

- Rollback of all tracked changes
- Work in an isolated fashion
- Seamless team collaboration
- Efficient and flexible scaleability

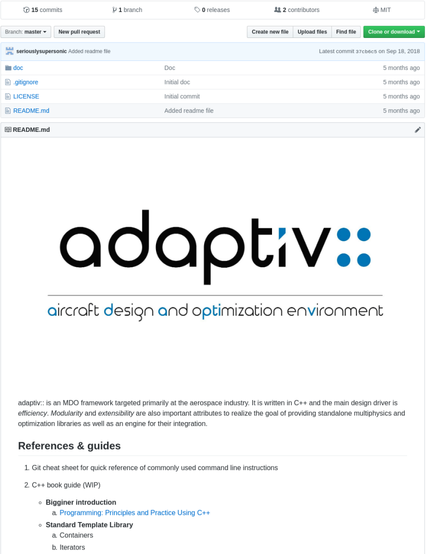
git - the world's leading version control system

Why git?

- Free and open-source
- Small and fast
- Encourages branching
- Distributed
- Built-in IDE support

As a service:

- Source code hosting
- Code sharing platform
- GitHub, GitLab, etc.



Choice of programming language

C++ (hard, lack of knowledge, modern features)
Use good well tested libraries (boost) - portability

WIP

Life cycle, architecture, design all depend on the requirements

Outline

Build system

TÉCNICO LISBOA

- Open-source, cross-platform set of tools to build, test and package software.
- Controls compilation process using platform and compiler independent config. files

CMake

The defacto standard for building C++ projects

Advantages

- More time for coding
- Supported by most popular IDEs (*e.g.* VS, JetBrains, QtCreator)
- Support for multiple compilers (*e.g.* MSVC, GCC, Clang, Intel)
- Easy integration of 3rd party libraries

Testing

content...

Benchmarking

content...

Outline

Outline

Interface elegance *vs* code efficiency

Level 3 BLAS operations $T(n) = O(n^3)$
Example:

$$C \leftarrow \alpha A^T B^T + \beta C \tag{1}$$

Desired interface:

```cpp
#include <matrix.hpp>

using DMatrix = Matrix<double>;

const double alfa = 42;
const double beta = 1.618

void example() {
    const std::size_t dim = 100;
    auto a = DMatrix::random(dim); // Same for b and c...

    c = alfa * a.transpose() * b.transpose() + beta * c;
}
```

Interface elegance *vs* code efficiency

For an efficient implementation, the statement

```
    c = alfa * a.transpose() * b.transpose() + beta * c;
```

should be translated into a call to the specialized CBLAS function:

```
    cblas_dgemm(CblasColMajor, CblasTrans, CblasTrans
               ,dim ,dim, dim
               ,alpha
               ,a.data(), dim
               ,b.data(), dim
               ,beta
               ,c.data(), dim);
```

Overhead:

1 function call
0 temporaries

Conventional operator overloading

Due to the normal order of evaluation of the C++ language,

```
    c = alfa * a.transpose() * b.transpose() + beta * c;
```

actually generates:

```
void example() { // Assume proper initialization of a, b and, c
    DMatrix temp1 = beta * c;         // call (A)
    DMatrix temp2 = b.transpose();    // call (B)
    DMatrix temp3 = a.transpose();    // call (B)
    DMatrix temp4 = temp3 * temp2;    // call (C)
    DMatrix temp5 = alfa * temp4;     // call (C)
    DMatrix temp6 = temp5 + temp1;    // call (D)
    c = temp6;                        // call (E)
}

DMatrix operator*(double d, const DMatrix& mat);           // (A)
void DMatrix::transpose();                                 // (B)
DMatrix operator*(const DMatrix& m1, const DMatrix& m2); // (C)
DMatrix operator+(const DMatrix& m1, const DMatrix& m2); // (D)
DMatrix operator=(const DMatrix& m1, const DMatrix& m2); // (E)
```

Conventional operator overloading

Overhead:

7+ function calls
6 temporaries

Performance comparison on a i7-4770k$_{(224\ \mathrm{GFLOPS})}$

TÉCNICO LISBOA

$$C \leftarrow \alpha A^T B^T + \beta C$$