1.
Optimizing software in C++
An optimization guide for Windows, Linux, and Mac platforms

By Agner Fog. Technical University of Denmark.
Copyright © 2004 - 2023. Last updated 2023-07-01.

Contents

1 Introduction ... 3
  1.1 Why software is often slow ... 4
  1.2 The costs of optimizing ... 4
2 Choosing the optimal platform ... 5
  2.1 Choice of hardware platform ... 5
  2.2 Choice of microprocessor ... 6
  2.3 Choice of operating system ... 7
  2.4 Choice of programming language ... 8
  2.5 Choice of compiler ... 10
  2.6 Choice of function libraries ... 11
  2.7 Choice of user interface framework ... 13
  2.8 Overcoming the drawbacks of the C++ language ... 14
3 Finding the biggest time consumers ... 15
  3.1 How much is a clock cycle? ... 15
  3.2 Use a profiler to find hot spots ... 16
  3.3 Program installation ... 18
  3.4 Automatic updates ... 18
  3.5 Program loading ... 18
  3.6 Dynamic linking and position-independent code ... 19
  3.7 File access ... 19
  3.8 System database ... 20
  3.9 Other databases ... 20
  3.10 Graphics ... 20
  3.11 Other system resources ... 20
  3.12 Network access ... 20
  3.13 Memory access ... 21
  3.14 Context switches ... 21
  3.15 Dependency chains ... 21
  3.16 Execution unit throughput ... 21
4 Performance and usability ... 22
5 Choosing the optimal algorithm ... 24
6 Development process ... 24
7 The efficiency of different C++ constructs ... 25
  7.1 Different kinds of variable storage ... 25
  7.2 Integers variables and operators ... 29
  7.3 Floating point variables and operators ... 31
  7.4 Enums ... 33
  7.5 Booleans ... 33
  7.6 Pointers and references ... 35
  7.7 Function pointers ... 37
  7.8 Member pointers ... 37
  7.9 Smart pointers ... 37
  7.10 Arrays ... 38
  7.11 Type conversions ... 40
  7.12 Branches and switch statements ... 43

7.13 Loops ... 45
7.14 Functions ... 47
7.15 Function parameters ... 50
7.16 Function return types ... 50
7.17 Function tail calls ... 51
7.18 Recursive functions ... 51
7.19 Structures and classes ... 52
7.20 Class data members (instance variables) ... 53
7.21 Class member functions (methods) ... 54
7.22 Virtual member functions ... 55
7.23 Runtime type identification (RTTI) ... 55
7.24 Inheritance ... 55
7.25 Constructors and destructors ... 56
7.26 Unions ... 57
7.27 Bitfields ... 57
7.28 Overloaded functions ... 58
7.29 Overloaded operators ... 58
7.30 Templates ... 58
7.31 Threads ... 61
7.32 Exceptions and error handling ... 62
7.33 Other cases of stack unwinding ... 66
7.34 Propagation of NAN and INF ... 66
7.35 Preprocessing directives ... 67
7.36 Namespaces ... 67
8 Optimizations in the compiler ... 67
8.1 How compilers optimize ... 67
8.2 Comparison of different compilers ... 76
8.3 Obstacles to optimization by compiler ... 80
8.4 Obstacles to optimization by CPU ... 85
8.5 Compiler optimization options ... 85
8.6 Optimization directives ... 87
8.7 Checking what the compiler does ... 88
9 Optimizing memory access ... 91
9.1 Caching of code and data ... 91
9.2 Cache organization ... 91
9.3 Functions that are used together should be stored together ... 92
9.4 Variables that are used together should be stored together ... 93
9.5 Alignment of data ... 94
9.6 Dynamic memory allocation ... 95
9.7 Data structures and container classes ... 97
9.8 Strings ... 105
9.9 Access data sequentially ... 105
9.10 Cache contentions in large data structures ... 106
9.11 Explicit cache control ... 108
10 Multithreading ... 110
10.1 Simultaneous multithreading ... 112
11 Out of order execution ... 113
12 Using vector operations ... 115
12.1 AVX instruction set and YMM registers ... 117
12.2 AVX512 instruction set and ZMM registers ... 117
12.3 Automatic vectorization ... 118
12.4 Using intrinsic functions ... 121
12.5 Using vector classes ... 125
12.6 Transforming serial code for vectorization ... 129
12.7 Mathematical functions for vectors ... 131
12.8 Aligning dynamically allocated memory ... 133
12.9 Aligning RGB video or 3-dimensional vectors ... 133
12.10 Conclusion ... 133