Improve run-time speed

(C) Richêl Bilderbeek

March 23, 2015

Chapter 1

Introductory questions

1.1 Question

- A programmer reports: 'I have found something inefficient and I have changed the code'
- What is the chance this changes run-time speed?
- What is the chance this improves run-time speed?

1.2 Answer

- What is the chance this changes run-time speed?
  - 20% or less
  - 80% of runtime is spent in 20% of the code[1]
- What is the chance this improves run-time speed?
  - Unknown, might be worse, might be better
  - Programmers have lousy intuition about run-time speed[2]

[1] Meyers, More Effective C++, item 16
[2] Meyers, More Effective C++, item 16

1.3 Question

```
//CLEVERGUY: I unwound the loop
//in this known-to-be speed-critical section
DoSomething(0);
DoSomething(1);
DoSomething(2);
//CLEVERGUY: Just look how inefficient this was B-)
//for (int i=0; i!=3; ++i) DoSomething(i);
```

1.4 Answer

- Compilers do a lot of optimizations for you
- You can assume loop unwinding is supported
- Correctness, simplicity, and clarity comes first[3]

[3] Sutter & Alexandrescu, C++ Coding Standards, Item 6

1.5 Question

```
class Phonebook
{
  //Assume always m_persons.size() == m_phonenumbers.size()
  std::vector<Person> m_persons;
  std::vector<int> m_phonenumbers;
  //...
};
```

1.6 Answer

```
class Phonebook
{
  //LUT = Look Up Table
  std::map<Person,int> m_lut;
};
```

1.7 Question

```
template <class ReturnType, class ContainerType>
const ReturnType Sum(const ContainerType& v)
{
  assert(v.size() != 0);
  //...
}
```

1.8 Answer

```
template <class ReturnType, class ContainerType>
const ReturnType Sum(const ContainerType& v)
{
  assert(!v.empty());
  //...
}
```

1.9 Question

```
struct MyClass
{
  MyClass(const std::string s)
  {
    m_s = s; sm_cnt++;
  }
  std::string m_s;
  static int sm_cnt;
};
```

1.10 Answer

```
struct MyClass
{
  MyClass(const std::string& s)
  : m_s(s)
  {
    ++sm_cnt;
  }
  std::string m_s;
  static int sm_cnt;
};
```

1.11 Question

```
int DivideByTen(const int x)
{
  int result = 0;
  asm
  {
     mov ecx,eax
     mov ecx, dword ptr[x]
     sar eax,1
     /* ... */
     add eax,ecx
     sar eax,3
     mov dword ptr[result], eax
  }
  return result;
}
```

1.12 Answer

```
int DivideByTen(const int x)
{
  return x / 10;
}
```

1.13 Question

```
//Re-use counters, instead of recreating them every time
int i = 0; int j = 0; const int sz = 10;

int main()
{
  for(i=0; i!=sz; ++i)
    for (j=0; j!=sz; ++j)
  //...
  for(i=0; i!=sz; ++i)
    for (j=0; j!=sz; ++j)
  //...
}
```

1.14 Answer

```
int main()
{
  const int sz = 10;

  for(int i=0; i!=sz; ++i)
    for (int j=0; j!=sz; ++j)
  //...
  for(int i=0; i!=sz; ++i)
    for (int j=0; j!=sz; ++j)
  //...
}
```

1.15 Question

```
int main()
{
  //Use my lightning-fast container
  MyContainer<MyClass> v = /* */;

  //Use my advanced sorting algorithm
  MySort(v);

  //Perform my lightning-fast copy
  MyContainer<MyClass> w = MyCopy(v);
}
```

1.16 Answer

```
int main()
{
  //Use a lightning-fast container
  std::vector<MyClass> v( /* */ );

  //Use an advanced sorting algorithm
  std::sort(v.begin(),v.end());

  //Perform a lightning-fast copy
  std::vector<MyClass> w(v);
}
```

1.17 Question

```
//MyClass cannot be default-constructed
std::vector<MyClass> v;
const int sz = 100000;
for (int i=0; i!=sz; ++i)
{
  v.push_back(MyClass(i));
}
```

1.18 Answer

```
//MyClass cannot be default-constructed
std::vector<MyClass> v;
const int sz = 100000;
v.reserve(sz);
for (int i=0; i!=sz; ++i)
{
  v.push_back(MyClass(i));
}
```

1.19 Question

```
const std::vector<int> GetLowestThree(std::vector<int> v)
{
  assert(v.size() >= 3)
  std::sort(v.begin(),v.end());
  v.resize(3);
  return v;
}
```

1.20 Answer

```
const std::vector<int> GetLowestThree(std::vector<int> v)
{
  assert(v.size() >= 3)
  //use std::partial_sort if lowest three must be sorted
  std::nth_element(v.begin(),v.begin() + 3, v.end());
  v.resize(3);
  return v;
}
```

1.21 Question

```
int main()
{
  CreateDatabase(); //Load a huge database

  std::cout << "Please enter an SQL statement\n";
  std::string s; std::cin >> s;

  //...
}
```

1.22 Answer

```
int main()
{
  std::thread t(CreateDatabase());

  std::cout << "Please enter an SQL statement\n";
  std::string s; std::cin >> s;

  t.join();

  //...
}
```

Chapter 2

Main advice

2.1 Overview 1/3

- Remember the 80-20 rule[1]
- Programmers have lousy intuition about run-time speed[2]
- Correctness, simplicity and clarity comes first[3]
- Know when and how to code for scalability[4]

[1] Meyers, More Effective C++, item 16
[2] Meyers, More Effective C++, item 16
[3] Sutter & Alexandrescu, C++ Coding Standards, Item 6
[4] Sutter & Alexandrescu, C++ Coding Standards, Item 7

2.2 Overview 2/3

- Don't optimize prematurely[5]
- Don't pessimize prematurely[6]
- Minimize global and shared data[7]
- Choose your library with care[8]
- Choose your container with care[9]

[5] Sutter & Alexandrescu, C++ Coding Standards, Item 8
[6] Sutter & Alexandrescu, C++ Coding Standards, Item 9
[7] Sutter & Alexandrescu, C++ Coding Standards, Item 10
[8] Meyers, More Effective C++, item 23
[9] Meyers, Effective STL, item 1

2.3 Overview 3/3

- Make copying cheap and correct for objects in containers[10]
- Choose your algorithm with care[11]
- Understand the legitimate use of custom allocaters[12]
- Consider concurrency[13]

[10] Meyers, Effective STL, item 2
[11] Meyers, Effective STL, items 31
[12] Meyers, Effective STL, items 31
[13] Sutter & Alexandrescu, C++ Coding Standards, Item 12

Chapter 3

Details

3.1 Big O

- Big O notation expresses the run-time cost depending on input size
- O(1): constant, e.g. accessing std::vector element
- O(n): linear, e.g. searching in an unsorted container
- O(log n): logarithmic, e.g. searching in a sorted container
- O(n^2): quadratic, e.g. insertion sort[1]

[1] `http://richelbilderbeek.nl/CppInsertionSort.htm`

3.2 Optimization techniques

- High[2]:
  - Improve critical algorithm its Big O
- Middle:
  - Newer compiler, newer C++ standard, newer library, concurrency
- Micro:
  - Profile guided optimization, inline, assembler

[2] Fuzzily ordered by yours truly

3.3 Algorithm to improve run-time speed

- Profile (in release mode!)
- Find bottleneck
- Think
- Change
- Profile
- Keep/undo change

3.4 Exercise

- Write an algorithm focussed on correctly doing X, e.g. determine if a number is prime
- Write one (or more) algorithm to do X faster
- Profile and determine which is fastest
- (check on multiple computers)

Chapter 4

EOF