Case Study for Yandex

Author: Ihar Marfin

Contact: <marfin@mail.desy.de>

Status: work in progress

Organization: DESY

Version: v0.1

Date: 03/28/2014

Dedication

For Yandex Company

Abstract

This document is a description of the tasks and corresponding solutions and answers. The solution in C++ is given.

Table of Contents

1 Preface

Answers to the question are supplied by the example codes. All together these codes are the project called case_study_yandex. The archive `case_study_yandex.tgz`, containing the project, is attached to this document. If it is not the case, please, download it from the Git repository.

The structure of the package is the following:

```
+-- bin
+-- docs
| +-- Case_Study_yandex.pdf
| +-- Case_Study_yandex.rst
| +-- Feature.cc.rst
| +-- Feature.h.rst
| +-- HOW-TO.rst
| +-- rst2pdf.style
| +-- Serialization.cc.rst
+-- features.dat
+-- featuresFixed.dat
+-- include
| +-- FeatureFixed.h
| +-- Feature.h
| +-- CpuTime.h
| +-- KahanSum.h
| +-- Serialization.cc
| +-- Serialization.h
| +-- Sum.h
+-- Makefile
+-- README
+-- src
    +-- Feature.cc
    +-- FeatureFixed.cc
    +-- KahanSum.cc
```

To compile, build and test the codes, please

```
make clean
make
```

It will produce executables in the `bin/` subfolder.

More details can be found in the Appendix [Makefile].

2 Question 1

2.1 Introduction to `Question 1`

- List the problems with the C++ code `here`. Refactor the code.

The code is the following [code]:

```
#ifndef FEATURE_H
#define FEATURE_H
```

```
#include <iostream>
#include <sstream>
#include <stdio.h>



class Feature
{
public:
    enum FeatureType {eUnknown, eCircle, eTriangle, eSquare};

    Feature() : type(eUnknown), points(0) { }


///Problem1
///     Feature( FeatureType ft ) : type(ft), points(0) { }

    ~Feature()
    {

///Problem2
        if (points)
            delete points;
//              delete [] points;
    }

    bool isValid()
    {
        return type != eUnknown;
    }

    bool read(FILE* file)
    {

///Problem6
//read first prt for points
///         fread(&points, sizeof(double *), 1, file);

///Problem4, Problem5 and Problem6

        if (fread(&type, sizeof(FeatureType), 1, file) != sizeof(FeatureType))
///         if (fread(&type, sizeof(FeatureType), 1, file) != 1)
            return false;
        short n = 0;
        switch (type)
        {
        case eCircle: n = 3; break;
        case eTriangle: n = 6; break;
        case eSquare: n = 8; break;
        default: type = eUnknown; return false;
        }
///Problem2 and Problem3
       points = new double[n];
```

```
        if (!points)
            return false;
///Problem6 and Problem7
        return fread(&points, sizeof(double), n, file) == n*sizeof(double);
    }
    void draw()
    {
        switch (type)
        {
/// Problem9
        case eCircle: drawCircle(points[0], points[1], points[2]); break;
/// Problem10
        case eTriangle: drawPolygon(points, 6); break;
        case eSquare: drawPolygon(points, 8); break;
        }
    }



protected:

/// Problem8
    void drawCircle(double centerX, double centerY, double radius);
/// Problem10
    void drawPolygon(double* points, int size);


///Problem1
    double* points;
    FeatureType type;
};


#endif
```

```
#include "Feature.h"
#include <iostream>


using namespace std;

int main(int argc, char* argv[])
{

/// Problem1

    Feature feature;
///  Feature feature(Feature::eSquare);

  FILE * pFile;
  pFile = fopen ("features.dat", "w");
  fwrite (&feature , sizeof(Feature), 1, pFile);
  fclose (pFile);
```

```
/// to test the Problem1
  feature.isValid();

    pFile = fopen("features.dat", "r");
    bool status=feature.read(pFile);
    fclose (pFile);

///Problem7
       if (!status) cout<<"something wrong with reading"<<endl;


    if (!feature.isValid()) {
        cout<<"feature:: invalid"<<endl;
        return 1; } else        cout<<"feature:: valid"<<endl;
    return 0;
}
```

2.2 Realization of `Question 1`

First, I consider that the task should be realized using C++. The package, called `case_study_yandex`, has been developed and tested with

```
gcc 4.6.1 in Ubuntu
```

To compile the the code, the `Make`-based framework is used. To build and setup executables, just do

```
make clean
make
```

To run the code, just run it as

```
bin/Feature
```

or

```
bin/FeatureFixed
```

The problematic code can be found in the `{src,include}/Feature.{cc,h}`. For my and your convenience, I have added comments, like `ProblemX`, where `X=1,2,3,etc`, to the code in order to indicate the problem. Also, the piece of the code

```
FILE * pFile;
pFile = fopen ("features.dat", "w");
fwrite (&feature , sizeof(Feature), 1, pFile);
fclose (pFile);
```

is added to test the problems bellow.

To solve the problem of I/O non-POD objects, the serialization of the `Feature` class is defined and tested in the `{src,include}/FeatureFixed.{cc,h}`.

2.2.1 The problem 1

- **Description**

The interface of the `Feature` class does not contain default functions to get/set its properties, i.e the members `points` and `type`. However, there is a special case of the "set" procedure, which is implemented as `bool read(FILE* file)`. This particular "set" procedure will take a significant amount of CPU due to file I/O operation, which makes the code to be slow.

- **Solution**

The interface of the `Feature` should be extended to include default `get()` and `set()`.

For example, `set()` function (here for `type` property) can be delegated to the constructor: `Feature( FeatureType ft ): type(ft) {}`.

Also setters of `type` and `points` members should be bound, in order to keep the correspondence between the `type` and the size of `points`.

2.2.2 The problem 2

- **Description**

This problem is related to the question: `Does "C++ delete array memory without brackets" works and how it works?` The C++ standard explicitly states that such behavior is undefined (undefined behavior, **UB**). Example of the **UB** for the different platforms is shown in Ref. [link1].

- **Solution**

Use brackets in the delete operator, `delete [] points`.

2.2.3 The problem 3

- **Description**

The `new` operator can throw the exception `std::bad_alloc`.

- **Solution**

Use `try {...} catch (std::bad_alloc& ba) {...}` block.

2.2.4 The problem 4

- **Description**

Relates to

```
if (fread(&type, sizeof(FeatureType), 1, file) != sizeof(FeatureType))
```

The `size_t fread (...)` returns the total number of elements successfully read [link2], but not the size of the type.

- **Solution**

Use, instead,

```
if (fread(&type, sizeof(FeatureType), 1, file) != 1)
```

2.2.5 The problem 5

- **Description**

Relates to

```
if (fread(&type, sizeof(FeatureType), 1, file) != sizeof(FeatureType))
```

The logic of the code is wrong. `Feature::read()` tries to read non-Plain-Object-Data (POD) from the file. The problem with objects of non-POD types is that once a copy of their memory storage is made (into a file using `fwrite()`), there's no such "tool" that we can use to do some operations with the copy. We can't `fread()` the copy back into memory and expect to use it as if it were the original object. More information can be found in Ref. [link3].

- **Solution**

I have solved this problem by using different logic of the `Feature::read()`. See, explanations later.

2.2.6 The problem 6

- **Description**

Relates to

```
if (fread(&type, sizeof(FeatureType), 1, file) != sizeof(FeatureType))
```

Anyway, the code works and sets 0, which is `Feature::eUnknown`, to the value of `type`. Why it happens? Because of the order of members in the class, this commands reads the field `points`, but not `type`. Occasionally, `sizeof(:Feature::FeatureType)` and `sizeof(double *)` are qualed, what prevent the code from the crash.

- **Solution**

To recover (partly) the reading problem of the non-POD, a few lines should be added before this `if(...)` statements

```
//read first prt for points
          fread(&points, sizeof(double *), 1, file);
```

2.2.7 The problem 7

- **Description**

Relates to

return fread(&points, sizeof(double), n, file) == n*sizeof(double);

As I pointed out, there is a wrong order of reading the fields from the files, first the `points` should be read then `type`. Another problem is that `points` is pointer. It contains the address which is not valid anymore after the serialization. So, this particular `fread()` will try to read `n doubles` from the record presenting the non-valid `u32` address to the `double` container. This `fread(...)` will return always 0. And then, if one checks the status of the reading operation in `main()`

```
bool status=feature.read(pFile);

.....
```

```
///Problem 7
      if (!status) cout<<"something wrong with reading"<<endl;
```

the warning will be always printed.

- **Solution**

To solve this problem, we have to implement the other logic of the `Feature::read()`. See, explanations later.

2.2.8 The problem 8

- **Description**

The `Feature::draw()` is based on two declared but not defined methods: `drawCircle()` and `drawPolygon`. Using the `Feature::draw()` in the main subroutine will cause the error of linking

```
Feature.h:76: undefined reference to `Feature::drawCircle(double, double, double)'
Feature.h:77: undefined reference to `Feature::drawPolygon(double*, int)'
Feature.h:78: undefined reference to `Feature::drawPolygon(double*, int)'
```

- **Solution**

Define the functions,

```
void drawCircle(double centerX, double centerY, double radius) {/* NOTHING-TO-DO */ }
void drawPolygon(double* points, int size) {/* NOTHING-TO-DO */}
```

2.2.9 The problem 9

- **Description**

The points are not properly initialized (they are always 0) in this call of `drawCicle(...)`

```
case eCircle: drawCircle(points[0], points[1], points[2]); break;
```

- **Solution**

Changing the logic of the `Feature::read()` (see, Problems 5-7) helps.

2.2.10 The problem 10

- **Description**

The wrong type of the first argument in

```
void drawPolygon(double* points, int size);
```

- **Solution**

Instead, use `const double *`, to prevent from modifying the content of the `points` pointer in this function.

```
void drawPolygon(const double* points, int size);
```

2.3 The `Feature` object serialization as a replacement of `Feature::read()`

The non-POD objects requires to have serialization/deserialization interface. I am not going to consider the `Boost::serialization` solution which handles the serialization of objects of any POD-complexity rather elegantly. Instead, I introduce the abstract layer `class Serializable`, which aims to perform the I/O operations with the members of the `class Feature`. For sake of simplicity, I do not follow the rules to achieve the cross-platform compatibility, i.e properly saving the little-endian and big-endians, which is discussed in details in Ref. [link3]. Therefore, the binary files, but not the plain ascii files, are chosen as storage elements.

The new interface with serialization support is called `class FeatureFixed`. The interface `Serializable` defines the two methods:

```
class Serializable
{

public:


   Serializable(){}
  ~Serializable(){}

   virtual void serialize  (std::ofstream& stream) {  }
   virtual void deserialize(std::ifstream& stream) {  }



};
```

The realization is rather simple and given in the `src/Serialization.cc`

```
#include "FeatureFixed.h"


#include <iostream>
#include <fstream>



void FeatureFixed::serialize (std::ofstream& stream){

        std::size_t size;
        if (type == FeatureFixed::eUnknown )  return;
        if (type == FeatureFixed::eCircle) size=3 ;
        else if  (type == FeatureFixed::eTriangle) size=6;
        else size=8;

        std::size_t type_size=sizeof(type);
        stream.write( reinterpret_cast<const char *>(&type), sizeof(FeatureFixed::FeatureType) );
        for (size_t i=0;i<size;i++) stream.write( reinterpret_cast<const char *>(&points[i]), sizeof(double));

}
```

```


void FeatureFixed::deserialize (std::ifstream& stream){

    std::size_t size;
```

```
    std::vector<char> buffer_type(sizeof(FeatureFixed::FeatureType));
    std::vector<char> buffer_point(sizeof(double));
    stream.read(&buffer_type[0], sizeof(FeatureFixed::FeatureType) );
    type = *reinterpret_cast<FeatureType *>(&buffer_type[0]);


        if (points) delete [] points;

        if (type == FeatureFixed::eUnknown ) { points=0; return; }
        else if  (type == FeatureFixed::eCircle )  size=3;
        else if  (type == FeatureFixed::eTriangle) size=6;
        else if  (type == FeatureFixed::eSquare)  size=8;
        else { type=FeatureFixed::eUnknown; points=0; return; }



        try { points = new double[size]; }  catch (std::bad_alloc& ba)
       {
            std::cerr << "bad_alloc caught: " << ba.what() << '\n';
            type=FeatureFixed::eUnknown; points=0; return;
        }


        for (size_t i=0;i<size;i++) {
            stream.read(&buffer_point[0], sizeof(double) );
            points[i] = *reinterpret_cast<double *>(&buffer_point[0]);

        }

        return;

}
```

The program, called `src/FeatureFixed.cc`, performs testing the serialization system of the `class FeatureFixed`.

3 Question 2

3.1 Introduction to `Question 2`

- For the three different implementations of floating point summation listed `here`, estimate the complexity (or performance) for the case, if size of input data array is increased by 1,000 times. Numbers in input data array are always positive. Please explain your answer and comment on the question.

```
double sum1(std::vector<double>& v)
{
   if (v.empty()) {
       return 0.0;
   }
   for(size_t i = 0; i < v.size() - 1; ++i) {
       std::sort(v.begin()+i, v.end());
       v[i+1] += v[i];
```

```
    }
    return v.back();
}

double sum2(std::vector<double>& v)
{
    if (v.empty()) {
        return 0.0;
    }
    for(size_t i = 0; i < v.size() - 1; ++i) {
        std::partial_sort(v.begin() + i, v.begin() + i + 2, v.end());
        v[i+1] += v[i];
    }
    return v.back();
}

double sum3(std::vector<double>& v)
{
    std::multiset<double> set(v.begin(), v.end());
    while (set.size() > 1) {
        std::multiset<double>::const_iterator itA = set.begin();
        std::multiset<double>::const_iterator itB = ++set.begin();
        double c = *itA + *itB;
        set.erase(itA, itB);
        set.insert(c);
    }
    return !set.empty() ? *set.begin()
                        : 0.0;
}
```

3.2 Answers on `Question 2`

I consider the performance of the algorithms using `big-O` notation and numbers reported in the tables of Ref. [link4]. Also, I appeal to the worst-case performance of the algorithms below. I assume that every sentence of the that doesn't depend on the size of the input data takes a constant `C` number computational steps.

- Let's consider all summation algorithms step by step.

For the first algorithm we have the computational complexity `f(N)` as

```
    if (v.empty()) {
        return 0.0;
    }
    for(size_t i = 0; i < v.size() - 1; ++i) {
        std::sort(v.begin()+i, v.end());
        v[i+1] += v[i];
    }
    return v.back();



f(N) =  C +  {N*Log(N) + (N-1)*Log(N-1) + ...+ 2*Log(2)}*C  +(N-1)*C  + C ~=
N + N^2*Log(N)
```

where `N*Log(N)` is the approximate complexity of the `std::sort`. In `big-O` notation the complexity of first algorithm is $O(N+N^2\ Log(N))$. So increasing the number of input elements by a factor of 1,000 corresponds to the increasing of CPU time by the factor of $\sim 7*10^6$.

The complexity of the second algorithm is the following:

```
double sum2(std::vector<double>& v)
{
   if (v.empty()) {
       return 0.0;
   }
   for(size_t i = 0; i < v.size() - 1; ++i) {
       std::partial_sort(v.begin() + i, v.begin() + i + 2, v.end());
       v[i+1] += v[i];
   }
   return v.back();
}


 f(N) =  C +  {N*Log(2) + (N-1)*Log(2) + ...+ 2*Log(2)}*C  +(N-1)*C  + C =
 N + (N-1)(N+2)/2*Log(2) ~= N + N^2/2
```

Here, following the Ref. [link5], the complexity of the `std::partial_sort` is approximately N*log(M), where N is size of the input vector, and M is the distance between first and middle elements, which is equaled to 2. In `big-O` notation the complexity of the second algorithm is $O(N+N^2\ /2)$. So increasing the number of input elements by a factor of 1,000 corresponds to the increasing of CPU time by the factor of $\sim 5*10^5$.

In the end, the calculation of the complexity for the third algorithm is estimated. I have found the error in the code. Instead of `set.erase(itA, itB);` `` should be ``set.erase(itA, ++itB);`. It reads

```
double sum3(std::vector<double>& v)
{
   std::multiset<double> set(v.begin(), v.end());
   while (set.size() > 1) {
       std::multiset<double>::const_iterator itA = set.begin();
       std::multiset<double>::const_iterator itB = ++set.begin();
       double c = *itA + *itB;
       set.erase(itA, ++itB);
       set.insert(c);
   }
   return !set.empty() ? *set.begin()
                       : 0.0;
}

f(N) = C + { (4 + log(N-1)) +  (4 + log(N-2)) + ... + (4 + log(1))    }*C  + C ~=
2*C + 4*(N-1)*C + (N-1)*Log(N-1) ~= 4*N + N*Log(N)
```

The `std::multiset` are `Red-Black Trees`. The complexity of `std::multiset::erase(it1,it2)` is proportional to the `it2-it1` [link6]. The time complexity of `std::multiset::insert(c)` is about of `O(log n)` [link7].

So increasing the number of input elements by a factor of 1,000 gives the increase of CPU time by the factor of $\sim 11*10^3$.

As a final examination, we can compare numerical results for each algorithm. I have realized the calculation of CPU cycles per each algorithm using functions from the GNU Standard Library `<time.h>` and `<sys/time.h>`. They are `gettimeofday(&time,NULL)` and `clock()`. More information can be found in `include/CpuTime.h`. Running the code `src/KahanSum.cc` [question3], we get the following numbers shown in the Figure 1.

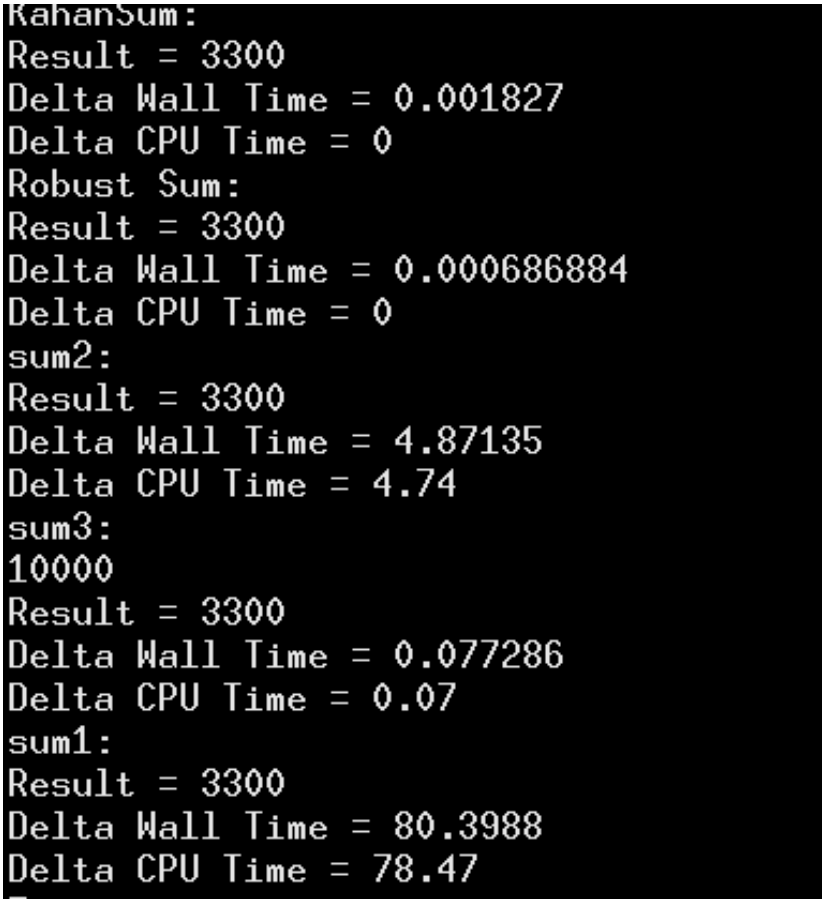

Figure 1. CPU timing for the summation different algorithms.

As expected, $CPU(sum1)/CPU(sum2) \sim 10$and $CPU(sum1)/CPU(sum3) \sim 10^3$. This absolutely agrees with the previous estimations.

4 Question 3

4.1 Introduction to `Question 3`

- Propose an optimal solution for task 2. Summation accuracy should remain the same.

4.2 Realization of `Question 3`

It is clear that the time complexity for the optimal summation is about `O(N)`. The fastest algorithms would be either

```
double sum3(std::vector<double>& v)
{
   if (v.empty()) {
```

```
        return 0.0;
    }
    for(size_t i = 0; i < v.size() - 1; ++i)
        v[i+1] += v[i];
    return v.back();
}
```

or

```
return std::accumulate(input.begin(), input.end(),0.);
```

However, the question about summation accuracy remains. To significantly reduce the numerical error of summation, I use the compensated summation aka Kahan summation algorithm [link8]. It reads as follows:

```
struct KahanResult
{
    double sum;
    double corr;
};


KahanResult KahanSum(KahanResult accumul, double val)
{
    KahanResult result;
    double input = val - accumul.corr;
    double sum = accumul.sum + input;
    result.corr = (sum - accumul.sum) - input;
    result.sum = sum;
    return result;
}



#include <iostream>
#include <cassert>
#include <vector>


int main(int argc, char* argv[])
{

    vector<double> input={0.01, 0.001, 0.0001, 0.000001, 0.00000000001};

  ///
  KahanResult init = {0.,0.};
  KahanResult result =
  std::accumulate(input.begin(), input.end(), init, KahanSum);

  std::cout << "Kahan Sum: " << result.sum << std::endl;
  std::cout << "Kahan Error: " << result.corr << std::endl;
  result 0;
```

```
}
```

The more details can be found in the `{interface,src}/KahanSum.{h,cc}`.

5 Question 4

5.1 Introduction to Question 4

- You are required to improve performance of an application written in C++. What are the typical performance bottlenecks? How can they be located and fixed?

5.2 Answers on Question 4

5.2.1 Introduction

1. C/ C++ applications are more tend to memory issues like memory leaks and memory corruption which could not only degrade the performance but also impact the scalability and availability of the applications and also, lead to unexpected results. The latter are even impossible to trace back. A vital care at the design of the memory management of C/C++ applications with the strict adherence to the usage of code review tools to identify the violation of coding standards can avoid this to a greater extent. These tools will be discussed later.
2. Next to memory it is the CPU timing. We know that the C/C++ application are less CPU intense since the code is available in the almost binary and, of course, optimized form that we have provided with the help of compilers, aimed to improve the efficiency of the code. If the code has its own drawbacks, we need to understand and validate the optimization level before we use it based on the solution we are working with.
3. General approach or tendency followed by the performance testers to improve the code is to identify the time taken by the functions and the number of function calls happening using in-built application tracer of some IDE or by using external run-time tools. Once the times are determined and the target is set, the algorithm itself is being modified using the strategy specified in the next subsection and methods of trails & errors.

5.2.2 Optimizing the C++ code

Before any optimization steps, the application is designed following two main rules:

1. The algorithms and data structures are designed in a way that makes sense for the application logic.
2. The code that implements the designed algorithms is written, following guidelines to avoid inefficient operations.

The points 1. and 2. are bound and results into suggestions that might be implemented in the code to reach high efficiency of the application. They are divided in few groups, discussed briefly below.

- **Optimization of the lifetime cycle**
  - We have to look into the perspective, trying to find such fragments `if (x != 0) x = 0;` and replacing it by `x=0`.
  - Then it is important to use inline functions everywhere it is possible.
  - It is typically more efficient to use `+=` instead of `+` alone, because we avoid generating a temporary object. We have to use the prefix operators to avoid the creating a temporary object.

- It is a good practice to use `explicit` constructors. Using `explicit` will protect you from stupid mistakes and make it easier for you to pinpoint potential bottlenecks.
- We have to prefer initialization over assignment for objects. Initializing an object invokes the object's copy constructor, while defining and then assigning the object invokes both the default constructor and then the assignment operator.
- The object declaration should be postponed until an initialization can be performed.
- The loops should be treated properly:
    a. LOOP UNROLLING
    b. LOOP JAMMING
    c. LOOP INVERSION
- An object should be passed to a function by reference, and not by pointer. This does not invoke a copy constructor.
- Also instead of passing a function pointer as an argument to a function, a pass of a function-object is more preferable.
- Any object constructor should initialize member objects using initialization lists.
- Two-phase construction of an object, based on the minimal constructor and the "heavy" factory, creating objects, will help in cases when many objects are copied.
- Exceptions are a great way to deal with unexpected errors. But they're expensive. They should be avoided.
- In every class, declare every member function that does not access the non-static members of the class as static . In other words, declare all the member functions that you can as static. In this way, the implicit this argument is not passed.

• **Memory optimization**

- One have to use the most simplest and thus efficient types: `int`, `unsigned int`, `char` etc.
- In order to save memory, we can turn off runtime type identification. `Type_info` objects will not waste the space. Thus `dynamic_cast` and `typeid` operators should be escaped.
- We can consider a class allocation by means of the "memory pools." If such allocators contain only methods ("empty classes") and they are used as a base class, then it can save more memory because of compiler byte-alignment settings.
- Using `copy-on-write` approach, two or more objects can share the same data until the moment when one of those objects is changed, at which point the data is physically copied and changed in one of the objects. This technique is realized with help of "smart" pointers.

• **Input/Output optimization**

- C++ stream I/O is a very flexible and safe method of doing input and output. However, the `printf` is more efficient, but not very safe.

• **Optimization of the sorting and searching**

- To search a sorted sequence, the `std::lower_bound`, `std::upper_bound`, `std::equal_range`, or `std::binary_search` generic algorithms should be used. Given that all the cited algorithms use a logarithmic complexity `O(log(n))` binary search, they are faster than the `std::find` algorithm, which uses a linear complexity `O(n)` sequential scan.

• **Optimization of the containers**

- The STL containers are efficient. There are a number of other advantages to using STL: It's a standard. It's thin. It's flexible. It's already written. And debugged, and tested.

- If a container provides a member function, that duplicates a generic STL algorithm, then the member function is more efficient.

3. Then, after design steps, the resulting software is tested against crippling defects. For this purpose, debuggers are used.
4. After having completed the development of a correctly working application, the **optimization stage** begins. Optimization is simply waste of programmer time if any of these statements are true:
   - parts of the program haven't been written yet;
   - the program is not fully tested and debugged;
   - it seems to run fast enough already;
5. The following sub-stages of the optimization are usually performed:

5.1. Performance testing (aka Profiling).

Commands with inadequate performance are detected. These are commands that, when processing typical data, require more resources (CPU time, storage space, etc) than are available. For every command with inadequate performance, a profiler is used to determine which portions of code are the so-called bottlenecks for that command. As an example about profiling, see next subsection.

Usually the profiling of the application, properly designed in accordance with 1) and 2) recommendations, will not show serious bottlenecks and further sub-stages are not required. However, because of the strong limits on CPU and Memory consumption, which are fixed in the software specification, the **Algorithmic optimization** is performed next. **Algorithmic optimization** is an extension of the design rules proposed for 1. and 2. steps.

5.2. Algorithmic optimization. For each bottleneck, optimization techniques are applied.

Such techniques can be found in algorithm theory textbooks. This optimization involves attempting to reduce the number of executed machine cycles. In particular it involves reducing the number of calls to costly routines or transforming expensive operations into equivalent but less costly operations.

5.3. Software platform dependent optimization.

For each bottleneck, optimization techniques are applied that depend upon both the programming language and the software platform. For example, compiler options, pragma compiler directives, language extensions, non-standard libraries, direct calls to the operating system are exploited. If this makes the program fast enough, the optimization stage is complete. Example: Loop unrolling with `gcc -funroll-loops` or function inlining with `gcc -finline-functions`.

5.4. Hardware platform dependent optimization. For each bottleneck,

optimization techniques that depend upon the hardware platform are applied. This can involve using machine instructions that are specific to a processor family.

The profiling step 5.1. of optimization, using the `perf` code, is shown in the next subsection.

5.2.3 Profiling of the C++ code

Here, as an example, a few standard tools `perf` [perf] and `groph2dot.py` [groph2dot] do profiling of the program `bin/KahanSum`. Running the commands

```
perf record -g -- ../bin/KahanSum
perf script | ../../gprof2dot.py -f perf -s -n0 -e0  | dot -Tpng -o output2.png
```

The `output2.png`, shown in Figure 2, gives detailed overview of the application performance

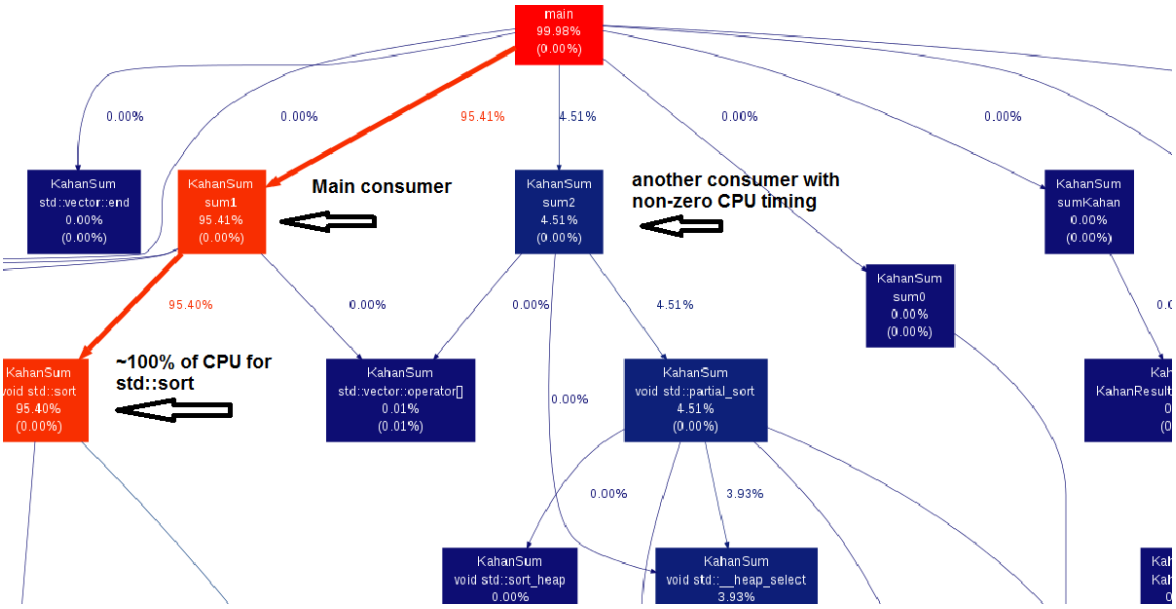

Figure 2. `perf` profiling of the `bin/KahanSum`.

The format of the plots is the following. A node in the output graph represents a function and has the following layout:

```
+------------------------------+
|        function name         |
| total time % ( self time % ) |
+------------------------------+
```

where:

- total time % is the percentage of the running time spent in this function and all its children;
- self time % is the percentage of the running time spent in this function alone;

An edge represents the calls between two functions and has the following layout:

```
          total time %
parent --------------------> children
```

where:

- total time % is the percentage of the running time transferred from the children to this parent (if available);

From Figure 2, the `sum1` and `sum2` consume over 99% of all CPU timing. The main consumers in both algorithms are `std::sort` and `std::partial_sort`. Certainly, the logic of the `sum1` and `sum2` should be changed to avoid use of the sorting algorithms.

6 Question 5

6.1 Introduction to Question 5

- Include random examples of your own C++ code that represent your skill level and your ability to design and implement your design.

6.2 Answers on `Question 5`

I'm a researcher working at the Deutschen-Electronen Synchrotron (DESY, Zeuthen/Hamburg) and Large Hadron Collider (LHC) of European Organization for Nuclear Research (CERN, Geneva).

1. I serve the development of the CMS detector, one of the experiments at LHC. Here I am one of co-conveners of the group developing the software for the CMS experiment. The software is called `CMSSW`. You can find the description and manual of it under the link `https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBook`. The source codes are available in the Git repository `https://github.com/cms-sw/cmssw`.
2. I carry out the statistical analysis of the LHC data and develop the software for statistical analysis within the CMS project. The software is called `HiggsAnalysis-CombinedLimit` and can be found in the Git:

```
git clone https://github.com/cms-analysis/HiggsAnalysis-CombinedLimit.git
```

3. I did once a case study for the company `HitFox GmbH`. You can find answers and corresponding sources in the Git repository:

```
git clone https://github.com/igormarfin/HitFox_Case_Study.git
```

More information can be found in `https://github.com/igormarfin/HitFox_Case_Study/blob/master/docs/Case_Study_hitfox.rst`.

4. Also as a freelancer, I contribute to the project called `reStructuredText`, which is the part of the python's Docutils. I develop different parsers within `rst2pdf`, `rst2latex` etc tools. Also I contribute to the `Pelikan` Static Site Generator, `http://blog.getpelican.com/`, developing the system of automated blog postings. An example of the system application is a test blog `http://igormarfin.github.io/pages/`.

7 Appendix

7.1 `Makefile` - the way to build and install the project

To build projects in C/C++, python and java and upload them to SVN and GIT repos, the `Makefile` is created. The following keywords, i.e `make keyword`, are supported:

```
all                          Commit2SVN                  info         makeSVN
clean                        DEPENDENCE.ProblemB_python  Makefile     rm_python_dirs
Commit2Git                   dirs2SVN                    makeGit
```

The Makefile is based on the use of hash tables (associative arrays) [makefile_assoc] and technique for processing multiple sub-folders with one Makefile [makefile_trick]. Each sub-project, either C/C++ or python, or JAVA, has two its own FLAGS (for compiling) and dependencies, like

```
#### Specific settings
DEPENDENCE.DivisorsHash                         :=
DEPENDENCE.ProblemA                    := $(INC_DIR)/DivisorsHash.h
DEPENDENCE.Node                                 := $(INC_DIR)/Node_Edge.cc
DEPENDENCE.Graph                       := $(INC_DIR)/Node_Edge.cc
DEPENDENCE.ProblemB                    := $(INC_DIR)/Node_Edge.cc

FLAGS.ProblemB                         := -DINT_MAX=100000

DEPENDENCE.ProblemB_python             := DEPENDENCE.ProblemB_python


DEPENDENCE.ProblemB_python:
        cd $(PYTHON_DIR) ; python setup.py test --test-suite='ProblemB.suite';
```

which are used to independently assemble the different sub-projects.

To build and setup executables, just do

```
make clean
make
```

Also, SVN and GIT repositories are supported. If user has registered to `googlecode.com` or `github.com`, it is possible to run the commands

```
# to creat SVN repo  and  upload the projects
make clean
make makeSVN
make dirs2SVN
sudo make Commit2SVN


# to create GIT repo and upload the projects
make clean
make makeGit
make Commit2Git
```

As an example, the `Makefile` can be defined as

```
#### Common settings

PROJECT                 := HitFox_Case_Study
TMPDIR = $$HOME/tmp/$(PROJECT)
TMPDIR2SVN = $$HOME/tmp/$(PROJECT)/SVN
CURDIR=${shell pwd }
Dirs            := ${shell find . -type d | grep -v ".svn" | grep  "./"}

### SVN repo settings
SVNREPO         := https://my-code-iggy-floyd-de.googlecode.com/svn/branches
SVNUSER                 := iggy.floyd.de@gmail.com

### GIT repo settings
GITREPOPATH     := https://api.github.com/user/repos
GITREPO         := $(PROJECT)
GITUSER         :=   igormarfin


### Compilers settings and C/C++ src/includes

CXX             := g++
CPPFLAGS        :=-g
SRC_DIR         := src
BIN_DIR                 := bin
INC_DIR                 := include


INCLUDES        := $(addprefix -I, $(INC_DIR))
SRC             := $(foreach sdir,$(SRC_DIR),$(wildcard $(sdir)/*.cc))
PROGS           := $(patsubst $(SRC_DIR)/%.cc,%,$(SRC))
HEADERS                 := $(patsubst $(SRC_DIR)/%.cc,$(INC_DIR)/%.h,$(SRC))
PROGS_IN_BIN    := $(patsubst src/%.cc,$(BIN_DIR)/%,$(SRC))



#### python programs settings



PYTHON_DIR      := scripts
PYTHON_INSTALL  := $(shell echo `pwd`/$(PYTHON_DIR)/test)

PYTHON_SRC      := $(shell ls -d $(PYTHON_DIR)/*/ | grep -v "build" | grep -v "dist" \
                  | grep -v $(PROJECT) | grep -v "test")
PYTHON_SRC      := $(patsubst $(PYTHON_DIR)/%,%,$(PYTHON_SRC))
PYTHON_SRC      := $(patsubst %/,%,$(PYTHON_SRC))

PYTHON_PROGS_IN_BIN     := $(addprefix $(BIN_DIR)/,$(patsubst %, %_python, $(PYTHON_SRC)))
PYTHON_PREFIX           := lib/python2.7/site-packages
```

```
VPATH           := $(SRC_DIR)  $(PYTHON_DIR) $(addprefix $(PYTHON_DIR)/, $(PYTHON_SRC))
FLAGS           := -std=c++0x



#### Specific settings  for C/C++ executables
DEPENDENCE.DivisorsHash                 :=
DEPENDENCE.ProblemA                 := $(INC_DIR)/DivisorsHash.h
DEPENDENCE.Node                         := $(INC_DIR)/Node_Edge.cc
DEPENDENCE.Graph                    := $(INC_DIR)/Node_Edge.cc
```

```
DEPENDENCE.ProblemB                 := $(INC_DIR)/Node_Edge.cc

FLAGS.ProblemB                      := -DINT_MAX=100000



all: rm_python_dirs   $(PROGS_IN_BIN) $(PYTHON_PROGS_IN_BIN)


#### Specific setting for python executables
DEPENDENCE.ProblemB_python          := DEPENDENCE.ProblemB_python


DEPENDENCE.ProblemB_python:
       cd $(PYTHON_DIR) ; python setup.py test --test-suite='ProblemB.suite';



#### To clear python directories

rm_python_dirs:
       - rm -r $(shell echo `pwd`/$(PYTHON_DIR))/dist
       - rm -r $(shell echo `pwd`/$(PYTHON_DIR))/build
       - rm -r $(shell echo `pwd`/$(PYTHON_DIR))/$(PROJECT).egg-info
       - rm -r $(shell echo `pwd`/$(PYTHON_DIR))/test


define make-goal
$(BIN_DIR)/$1: $2 $3 ${DEPENDENCE.${1}}
       $(CXX) $(INCLUDES) $(CPPFLAGS) $(FLAGS)  ${FLAGS.${1}} $$< ${DEPENDENCE.${1}}   -o $$@
endef



define make-goal-python
$(BIN_DIR)/$1: $2 ${DEPENDENCE.${1}}
       cd $(PYTHON_DIR);  python setup.py bdist_egg;
       mkdir -p $(PYTHON_INSTALL)/$(PYTHON_PREFIX)
       sudo  sh -c "export PYTHONPATH=$(PYTHON_INSTALL)/$(PYTHON_PREFIX):${PYTHONPATH};\
       easy_install --install-dir=$(PYTHON_INSTALL)/$(PYTHON_PREFIX)\
       --script-dir=$(shell echo `pwd`/$(BIN_DIR)) $(shell echo `pwd`/$(PYTHON_DIR)/dist/$(PROJECT)\
       -0.1-py2.7.egg)"
       echo export PYTHONPATH=$(PYTHON_INSTALL)/$(PYTHON_PREFIX):${PYTHONPATH}
endef


info:
       $(info  SOURCES:         $(SRC))
       $(info  PROGRAMS:        $(PROGS))
       $(info  INCLUDES:        $(INCLUDES))
       $(info  HEADERS:         $(HEADERS))
       $(info  PYTHON_SRC:      $(PYTHON_SRC))
       $(info  PYTHON_INSTALL:  $(PYTHON_INSTALL))
```

```
       $(info  PYTHON_PROGS_IN_BIN:     $(PYTHON_PROGS_IN_BIN))


clean: rm_python_dirs
       - rm $(BIN_DIR)/*


$(foreach prog,$(PYTHON_SRC),$(eval $(call make-goal-python,$(prog)_python,\
                  $(PYTHON_DIR)/$(prog)/$(prog).py )))
$(foreach prog,$(PROGS),$(eval $(call make-goal,$(prog),$(SRC_DIR)/$(prog).cc,\
                  $(INC_DIR)/$(prog).h )))
```

```
###SVN support
makeSVN:
        svn mkdir  $(SVNREPO)/$(PROJECT) --username $(SVNUSER) ;\

dirs2SVN:
       - for dir in $(Dirs); do \
       svn mkdir  $(SVNREPO)/$(PROJECT)/$${dir} --username $(SVNUSER) "adding $${dir}" ; \
       done;



Commit2SVN:
        mkdir -p $(TMPDIR2SVN); \
        cd  $(TMPDIR2SVN) ; \
        svn co $(SVNREPO)/$(PROJECT); \
     ls ; \
     cd - ; \
     find  ./ -iname "*" | grep -v "*.svn*" | xargs -I {} install -D {}\
     $(TMPDIR2SVN)/$(PROJECT)/{} ; \
     cd  $(TMPDIR2SVN)/$(PROJECT)  \
        ls ;\
        pwd ;\
        svn status | grep '?'   | sed 's/^.* /svn add --parents --force\
        --username $(SVNUSER) /' | bash ; \
        svn ci ; \
        cd $(CURDIR) ;\
        rm -r  $(TMPDIR2SVN);


##########Git support
makeGit:
       curl -u '$(GITUSER)' $(GITREPOPATH)  -d '{"name":"$(GITREPO)"}'

Commit2Git:
       git init
       git add ./
       git commit -m "adding to GitHub.com"
       git remote add origin git@github.com:$(GITUSER)/$(GITREPO).git
       git push -u origin master

.PHONY: all  info clean rm_python_dirs   makeSVN dirs2SVN Commit2SVN
```

8 References

code	The code snips are obtained using the following shell command
	`cat include/Feature.h | awk '{printf "  %s\n",$0}' > Feature.h.rst`
	`cat src/Feature.cc | awk '{printf "  %s\n",$0}' > Feature.cc.rst`

link1	Is delete[] equal to delete? http://stackoverflow.com/questions/1553382/is-delete-equal-to-delete
link2	fread http://www.cplusplus.com/reference/cstdio/fread/
link3	Disch's tutorial to good binary files http://www.cplusplus.com/articles/DzywvCM9/
link4	Computational performance http://en.wikipedia.org/wiki/Best,_worst_and_average_case
link5	`std::partial_sort` http://www.cplusplus.com/reference/algorithm/partial_sort/
link6	`std::multiset::erase` http://www.cplusplus.com/reference/set/multiset/erase/
link7	`std::multiset::insert` http://www.cplusplus.com/reference/set/multiset/insert/
link8	Kahan summation algorithm http://en.wikipedia.org/wiki/Kahan_summation_algorithm
perf	perf Examples. How to profile your applications using the Linux perf tools http://www.brendangregg.com/perf.html http://baptiste-wicht.com/posts/2011/07/profile-applications-linux-perf-tools.html
groph2dot	Gprof2Dot http://code.google.com/p/jrfonseca/wiki/Gprof2Dot#Example
makefile_assoc	Makefile: find in array. http://stackoverflow.com/questions/7282414/makefile-find-in-array
makefile_trick	How to generate a Makefile with source in sub-directories using just one makefile http://stackoverflow.com/questions/231229/how-to-generate-a-makefile-with-source-in-sub-directories-using-just-one-makefil