DV1567 - Project 2
Performance and scalability analysis of a C application

The goal of the project is to make your group to play the double role of the performance engineering team and DevOps team. As DevOps team, you develop and deploy a very simple application named ReSoW (guidelines are given in Annex 1). As performance engineering team, you conduct performance and scalability testing of ReSoW and you will produce a bottlenecks analysis and guidelines to improve the performance and scalability of the application. The DevOps team use the results from the performance and scalability test to improve the software performances and scalability, that will be tested again by the Performance Engineering team.

ReSoW is a simple application that read data from the disk, process the data (sorting) and then write back the data on the disk. ReSoW is written in C. ReSoW can be classified a batch job.

In what follow are described: the ReSoW application, the performance test scenario, the guidelines for performance optimization, the scenarios for the scalability test and analysis, and finally, what should be the content of the project report.

The ReSoW application

In what follow is provided the pseudocode of the ReSoW application. Guidelines and pseudocode for implementing the three functions are provided in Annex 1.
The suggested buffer size is 102400 records (numbers). This is an indicative value and depend on the characteristics of the HW you are using (CPU and memory capacity, disk speed).

ReSoW pseudocode

```
Input: DataSetSize, BufferSize, DatasetFilename, OutputFilename
Output: the file OutputFilename containing the sorted dataset.

main {
   // load the dateset in the memory area addressed by ds
   ds = loadDataset(dataset,DataSetSize,Buffersize);
   // compute the average value of the dataset, i.e. sum_of_dataset_values / num_of_dataset_values
   avg = average(ds)
   // find the max value in the dataset
   max = maxvalue(ds)
   // find the min value in the dataset
   min = minvalue(ds)
   //sort the dataset and copy it into the memory area pointed by sds
   sds = sortDataset(ds,sortingAlgorithm);
   //write the sorted array into a new file plus the valies of the average, min and max as the first three records.
   writeDataset(OutputFilename,sds,Buffersize, avg, min, max)
}
```

Performance test scenarios

To collect the baseline performance counters, you should run a single instance of the ReSoW application as follow

- Sorting algorithm = insertion sort
- Standard read/write buffer size
- Read/write block size equal to the size of a single record in the file, that is a floating point number.
- gcc compiler optimization turned off

The function to create the dataset should not be included in the measurement and in the performance and scalability analysis. It is enough to create the dataset only once.

From the collected data, you will discover the hotspot and then you can proceed with the performance and scalability optimization.

Performance optimization

Because you can't change the underlining HW and SW platform the only option is to optimize the code. You can use a feaster sorting algorithm, e.g. quick sort (qsort function in C), you can optimize the I/O operations playing with different buffer size and with the size of block read/written, you can apply loops, branch and memory optizations. You can optimize the code by yourself or you can rely on the gcc compiler optimization.

Apply at least two optimization techniques.

Collect the performance counter again and eventually iterate the optimization process.

Scalability test and analysis

You could analyse the application scalability by means of new set of experiments. The scalability should be analysed with respect to dataset size and buffer size:

- First, run an instance of ReSoW and increase the size of the dataset multiplying by a factor N the original dataset size, e.g. N=10, 100, 1000, … You can choose a different increment step depending on the baseline performance measurement collected and depending on the characteristics of the HW you are using.
- Second, keep constant the dataset size (e.g. N=10240000) and change the buffer size, e.g. buf_size = 128, 512, 1024, 2048, …

NOTE: in case in the scalability analysis you observe new hotspot or you think the application can be further optimized, you should iterate the optimization process.

Final report

The project report should contain:

- The performance test results before the optimization
- The hot spot analysis

- The guidelines for performance improvement
- The description of the optimizations implemented (and eventually the many optimization iteration you perform)
- The performance test results after the optimization (eventually for each optimization iteration)
- The scalability test/analysis results
- The non optimized code (provide in a separate file)
- The optimized code if completely hand-made, otherwise the gcc optimization options used, or both if a mixed approach is used (provide in a separate file)

Annex 1

In what follow are suggested possible implementations of the main software components of the ReSoW application. However, you are free to choose different implementations.

Guidelines for implementing loadDataset (pseudocode)

```
//this code is an example of how to read a file in C
#include<stdio.h>

int loadDataset (int DataSetSize, char *filename) {
  FILE *fp;
  float *v;
  fp = fopen( filename , "r" );
  for( i=1; i<=DataSetSize, i++){
              fread(v[i] , 1 , sizeof(float) , fp );
  }
  // TIP you can define you own buffer, buffer size and you can read blocks of data of size > 1
  fclose(fp);
}
```

Guidelines for implementing sortDataset as Selection Sort.

```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Guidelines for implementing writeDataset (pseudocode)

```
#include<stdio.h>
int writeDataset (int DataSetSize, char *filename)
{
   FILE *fp;
   float *v;
   …
   fp = fopen( filename , "w" );
   for( i=1; i<=DataSetSize, i++){
              fwrite(v[i] , 1 , sizeof(float) , fp );
   }
  // TIP you can define you own buffer, buffer size and you can write blocks of data of size > 1
   fclose(fp);
   return(0);
}
```

Pseudocode for creating the dataset

```
Input: DataSetSize, DatasetFilename;
Output: the file DatasetFilename containing the dataset

      float v[DataSetSize];
      //you can alternatively dynamically allocate the vector
      for i = 1 to DataSetSize {
                  v[i]=generateRand(100); }
      //create a floating point random number between 1 and 100
      end
      f=open("DatasetFilename","w")
      e=fwrite(f,sizeof(float),DataSetSize,v);
      fclose(f);
```

Pseudocode for generating a random number in C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

float generateRand(int rmax) {
  //Generate a floating point random number between 0 and rmax
  srand((unsigned int)time(NULL)); //initialize the random number generator
  return ( (float)rand()/(float)(RAND_MAX)) * rmax );
}
```

Pseudocode for computing the average, max and min values

```
// the code is trivial you don't need any specific guideline!!!!!
```