Big Data Analytics Programming
Assignment 1: *Blocked Matrix Multiplication and Bitmaps*

Pieter Robberechts
pieter.robberechts@cs.kuleuven.be

Thomas Dierckx
thomas.dierckx@cs.kuleuven.be

Laurens Devos
laurens.devos@cs.kuleuven.be

Collaboration policy:
Projects are independent: no working together! You must come up with how to solve the problem independently. Do not discuss specifics of how you structure your solution, etc. You cannot share solution ideas, pseudocode, code, reports, etc.

For computing the frequent word pairs, you may use (but do not necessarily have to):

`https://github.com/WojciechMula/sse-popcount`

If you use one of these algorithms, you must properly acknowledge it in your report. Beyond what is provided, you may not use any other code that is available online. You cannot look up answers to the problems online. If you are unsure about the policy, ask the professor in charge or the TAs.

1 Blocked Matrix Multiplication [4 points]

In this part of the assignment you will implement blocked matrix multiplication in C. Look at matrix_multiplication.c and complete the parts marked `TODO`. For simplicity, we will work with square matrices. You can use `gcc` to compile the code:

```
$> gcc matrix_multiplication.c -o matrix_multiplication
```

Make a script "mulmatrix.py". The first argument to the script is a file name F. The remaining arguments are possible block sizes. Each line in F will be of the form

`<N><whitespace><filename_matrixA><whitespace><filename_matrixB>`.

In this format, `<N>` is size of the matrix (N x N) and `<filename_matrixA>` and `<filename_matrixB>` are the matrices to be multiplied. The file "input_matrix.txt" contains an example of the file format.

The script should call your matrix multiplication code to multiply matrixA and matrixB using the naive strategy and the blocked strategy for each given block size.

Then the script should create a graph 'graph.png' that shows the results with matrix size on the x-axis and time on the y-axis. Create one line for each block size and one line for the naive strategy. This plot should be included in your report.

2 Finding Frequent Word Pairs [8 points]

In this part of the assignment, you will implement a C program that reports the number of times a pair of words appears together in a document for pairs that appear at least some set *threshold* number of times. The following files are provided:

- `Makefile`: Use `make` in the source folder to build the code. Use `make clean` to remove all build files.
- `main.c`: The file containing the `main` function, which implements the command line interface.
- `dataset.*`: These files contain the data-loading logic and define the `dataset` object, which contains two representations of a *bag-of-words* encoding of a collection of documents. The first representation is a *list-of-indexes* representation: for each word in the vocabulary, we store a list containing the *sorted* identifiers of documents that contain this word. The second representation is a *bitmap* representation: for each word in the vocabulary, we construct a bitmap storing ones for documents that contain the word, and zeros for documents that do not contain the word. `dataset.h` has two methods to access the respective representations: `get_term_indexes` and `get_term_bitmap`.
- `output.*`: These files define how you output your results. Use the `push_output_pair` function to output frequent pairs, as illustrated by the example functions `find_pairs_naive_bitmaps` and `find_pairs_naive_indexes` in `find_frequent_pairs.c`.
- `find_frequent_pairs.c`: This file contains **the two functions that you have to implement**: `find_pairs_quick_indexes` and `find_pairs_quick_bitmaps`. Both functions should produce the same output, but the first should use the *list-of-indexes* representation, and the second should use the *bitmap* representation. You are graded on the single-core speed of your implementations, so try to make the implementations as fast as possible.[1]

  The signature of the two functions is the same:

  ```
  find_pairs_quick_*(const dataset *ds,
                     output_pairs *op,
                     int threshold);
  ```

  The dataset object gives you access to the data using the `get_term_indexes` and `get_term_bitmaps` functions (`dataset.h`). The `output_pairs` object lets you efficiently output the frequent pairs you found using `pus_output_pair`. You should discard all pairs that appears less frequently than the given *threshold*, reporting them is a mistake.

The data file for this exercise is `bow.dtm.gz` (or `bow.dtm.zip`) and is accessible from Toledo. This is a simple compressed ASCII data file containing information about which words are used in which documents. The uncompressed file is large (about 600MB) compared to the compressed file (about 10MB). You can choose to decompress the data and work with the text file directly as follows:

```
./find_frequent_pairs <..>/bow.dtm THRESHOLD METHOD NUM_RUNS
```

or you can work with the compressed data directly:

[1] Don't forget about one very important optimization, see exercise session 1. Hint: frequent item set mining.

```
gzip -dc <..>/bow.dtm.gz | ./find_frequent_pairs - THRESHOLD METHOD NUM_RUNS
```

For testing purposes, we have also included a smaller bag-of-words file: `small.dtm`. Possible values for `METHOD` are `naive_indexes`, `naive_bitmaps`, `quick_indexes`, and `quick_bitmaps`. The latter two call your respective implementations, the former call the naive example implementations.

3 Report [3 pts]

The report should be a maximum of one page, excluding figures.

First, remember to include the graph for the matrix multiplication experiment in the report.

Second, the goal of the bitmap exercise is to experiment with the two data representations. Which of the two representations allows for the fastest algorithms? How do you optimally implement these algorithms? Are there circumstances in which the other representation is better? Could you improve the data representations?

If you have used code from `https://github.com/WojciechMula/sse-popcount`. If you use one of these algorithms, remember to acknowledge it in your report and add one sentence justifying your choice.

4 Important remarks

Deadline The assignment should be handed in on Toledo before Tuesday November 12 at 9am. There will be a 10% penalty per day, starting from the due day. The project can be a maximum of three days late: no projects submitted after November 15th at 9am will be graded.

Deliverables You must upload an archive (`.zip` or `.tar.gz`) containing the report (as pdf) and all the source files (see file hierarchy below). Note that for the second part of the assignment, you should only include `find_frequent_pairs.c`, that is, you should not modify any of the other files.

```
assignment1/
├── report.pdf
├── MatrixMultiplication
│   ├── matrix_multiplication.c
│   └── mulmatrix.py
└── FrequentPairs
    └── find_frequent_pairs.c
```

- **Do not** upload any data or output files.

Evaluation Your solution will be evaluated based on the correctness of your results and the speed of your code.

Automated grading Automated tests are used for grading. Therefore you must follow these instructions:

- Use the exact filenames as specified in the assignment.

- Do not change the command line arguments of the matrix_multiplication.c script.
- For the second part of the assignment, all your code should be in `find_frequent_pairs.c`. Do not add additional files. Do not modify any of the other files. You should only use the C standard library, no additional libraries are allowed. You cannot use `pthreads`, for example, so you should **focus on single-core performance**.
- Your code must compile and run on the departmental machines, this can be done over ssh (see documents of exercise session 1).

Running experiments This is a class about big data, so running experiments could take hours or even multiple days. You can do this easily on the departmental machines: it takes several minutes to start the experiments and then you check back later to see the results. Failure to run and report results on the full data set will result in a substantial point reduction.

Good luck!