W4112 Database Systems Implementation
Spring 2022
Project 2

In this project, you are asked to implement in C algorithms for various versions of binary search and to use them to implement an in-memory band-join algorithm. Grading will be based on both your code, and your analysis of the performance results you generate on Google cloud machines. You will work in groups of two or three. For three person groups, there is an additional programming requirement.

By default we'll be using the same groups as for project 1. If your partner from project 1 dropped the class, or if you wish to work with different people this time, the TAs will facilitate the matching of students. The TAs will post details of the matching process on Ed. If for some reason (e.g., a partner drops the class well into the semester) a student ends up working alone, the student is expected to complete the entire project expectations for 2-person groups.

The project consists of several parts, each with its own submission deadline. You can download the code templates from courseworks under the "Files" tab. You will be running the code on Google cloud machines. The TAs will provide each of you with a $50 coupon for the use of the Google cloud. The TAs will separately provide documentation for how to use the Google cloud for this project. In particular, it will be important for you to select the right class of machines that support the full AVX512 SIMD instruction set.

Details of exactly how to submit the code and pdf files will be provided later.

Part 1: Data-Dependent Binary Search

(Due 4/13/22. 50 points)

In this part of the project, you will explore the conversion of control dependencies to data dependencies to improve performance. You are supplied with a fully functional C function `lower_bound` that performs a binary search where you want the the index of the first key that is bigger than or equal to the search key. You are initially expected to write two C functions `lower_bound_nb_arithmetic` and `lower_bound_nb_mask` that replace the `if` test in `lower_bound` with code that does not use either the `if` statement or the C idiom `condition ? x : y`. As implied by the names of the functions, `lower_bound_nb_arithmetic` should use arithmetic operations to avoid `if` statements, while `lower_bound_nb_mask` should use bit manipulation operations (e.g., `&,|,~`) and masking. *Hint: you may use one subtraction in the* `lower_bound_nb_mask` *function, and the 64-bit representation of -1 is 0xFFFFFFFFFFFFFFFF.*

You should now write a function `lower_bound_nb_mask_8x` that tries to do 8 binary searches in parallel, hoping that the underlying system will overlap the cache miss latencies for concurrent searches. Inside this routine, there should be an outer `while` loop to check a termination condition, and an inner `for` loop to iterate through each of the 8 searches. The inner loop should use the same mask-based computation as your `lower_bound_nb_mask` algorithm. These is a subtlety in this parallelization in that some of the 8 searches may need more iterations than others. In such a case your loop should continue until all searches are complete and should make sure that "extra" loops of a completed search do not change the search result.

Finally, you should write a function `lower_bound_nb_mask_8x_AVX512` that uses AVX512 SIMD instructions to do 8 binary searches in parallel. In AVX512, 512-bit registers can hold eight 64-bit values and operate on the component values in parallel. You can find a set of functions called "intrinsics" that will be useful for the project here. For our purposes, you will primarily need functions that have a `_mm512` prefix and a suffix `_epi64` meaning that they operate on 64-bit signed integers. Within these instructions, you'll probably want to use some combination of the following types of instructions:

setl	These instructions allow you to load all elements of a vector with a scalar value.
cmp..	These are various comparison instructions such as `cmpge` and `cmplt` that compare two registers and set a scalar bitmap as the result.
srli	Shift right. Handy for dividing by 2.
mask_blend	Combine data from two different registers into a third register depending on the mask bitmap.
i64gather	Load 8 values from 8 different offsets from a base address.
add,sub,...	There are various arithmetic functions available.

You will profile the code using the provided `bulk_binary_search` and `bulk_binary_search_8x` functions. Simply uncomment the variant of the code that you want to measure within `bulk_binary_search` and `bulk_binary_search_8x`. You will run the code by invoking

```
./db4112 N X Y Z R
```

For this part of the project you can ignore `X`, `Y` and `Z` and supply dummy values. `N` is the number of elements in the array being searched, which is also the same as the number of probes. For this test, the provided code searches the array for each of its elements in a random order. `R` is the number of repeats of the experiment within the timing loop, which you should set to an appropriate number to deliver stable results. (If this number is too low, then various overheads of setting up the search may dominate the measured time; if this number is too high, your experiment will take too long.) For example, when I run

```
./db4112 10000 1 1 1 1000
```

to profile `lower_bound` on a local machine, the output is

```
Time in bulk_binary_search loop is 1016150 microseconds or 0.101615 microseconds per search
Time in bulk_binary_search_8x loop is ...
```

For this part of the project you will submit your code for the `lower_bound_nb_arithmetic`, `lower_bound_nb_mask`, `lower_bound_nb_mask_8x` and `lower_bound_nb_mask_8x_AVX512` functions. You will also submit a pdf document containing a graph showing the measured performance of the five routines as you varied `N` from 10 to 10^7. So `N` should be on the x-axis in log-scale, time-per-search should be on the y-axis in linear-scale, and your graph should have five labeled curves, one per function. Make sure to measure enough `N` values to generate reasonable resolution in your results. In the pdf document, you should:

1. State the machine characteristics for the machine on the Google cloud that you used. See the companion Google cloud documentation for how to identify the machine characteristics.
2. State which of the methods performs best in your experiments.
3. In one to two paragraphs, explain the shape of your measured performance curves. For example, you might notice an increase in the time taken at particular values of `N`. Explain the significance of these particular `N` values.

Part 2: Band Join

(Due 4/25/22. 30 points for 2-person groups; 50 points for 3-person groups)

In this part of the project, you will use the `lower_bound_nb_mask_8x_AVX512` method you previously developed to implement an in-memory band join.[1] Band-joins are useful when the join predicate involves inequalities. For example, consider the SQL query

```
SELECT *
FROM Jobs J, Candidates C
WHERE J.salary >= C.request-10000
  AND J.salary <= C.request+10000
```

Suppose we sort the candidates by their `request` values. Then we can use binary search to find the first possible candidate for a job by searching for `J.salary-10000`. We can then scan through the ordered candidates one by one, outputing each as a match with the `Jobs` record, until we find a candidate with `request` $>$ `J.salary+10000`.

For this part of the project, you will implement the `band_join` function according to the algorithm outlined above. As mentioned in the code template, you need to make sure you don't exceed the size of the array allocated for the output results. If you hit that size limit, you should stop there, and output what you have so far, which should be a prefix of the full band join results. Your algorithm should process 8 records at a time where possible using `lower_bound_nb_mask_8x_AVX512`. When the outer table cardinality is not a multiple of 8, there will be a few leftover records that you should handle with `lower_bound_nb_mask`.

[1] In the event that you did not successfully complete Part 1, we will supply code that you can use for `lower_bound_nb_mask_8x_AVX512` and `lower_bound_nb_mask` four days after the Part 1 deadline.

Three person groups. One performance overhead of the previous `band_join` implementation is that one has to test the search key for every inner table record. Three-person groups should also implement `band_join_opt`, which avoids testing the search key in the inner loop. See the code template for more details.

Profiling the code involves running

```
./db4112 N X Y Z R
```

as before. For band-join, the `R` value is ignored, and can be replaced with a dummy value or omitted. `N` is the size of the inner (sorted) table of the band-join, and `X` is the size of the outer table. `Y` is the size of the output result being allocated, and `Z` is the bound being used in the band computation. In other words, the outer key must be within `Z` (inclusive) of the inner table value. The outer table values are generated by the system as 31-bit random numbers that are independent of the values in the inner table. So, for small `Z` values, you probably won't get any matches.

For this part of the project you will submit your code for `band_join` and (where appropriate) `band_join_opt`. You will also submit a pdf document containing a graph showing the measured performance of the band-join routines. Pick an interesting value for `N` and `X` and justify in a sentence or two why you think they are interesting. Choose a `Y` value that is large enough to hold the results, except perhaps for the very largest join results, but not so large that you exceed the physical RAM of the Google cloud machine that you're using. For these particular choices, vary `Z` to get a range of output sizes. In your graph `Z` should be on the x-axis, time-per-output-record should be on the y-axis, and your graph should have one or two labeled curves, one per function. In the pdf document, you should:

1. State the machine characteristics for the machine on the Google cloud that you used. See the companion Google cloud documentation for how to identify the machine characteristics.

2. In one to two paragraphs, explain the shape of your measured performance curve(s). For example, you might notice a change in the time taken at particular values of `Z`. Explain the significance of these particular `Z` values.

Grading

Two-person groups will be graded out of 80, while 3-person groups will be graded out of 100. Two-person groups may attempt the additional `band_join_opt` part if they wish, and their grade will be computed as the maximum of (a) the grade for the 2-person project without this part, out of 80 and (b) the grade they would have received under the 3-person grading scheme, out of 100.