Throughout the report, when an input size is specified as 100, it means that the matrix multiplication was done with 2 matrices, each is in the shape of 100 by 100. For brevity, all references to input size will be implied as such.

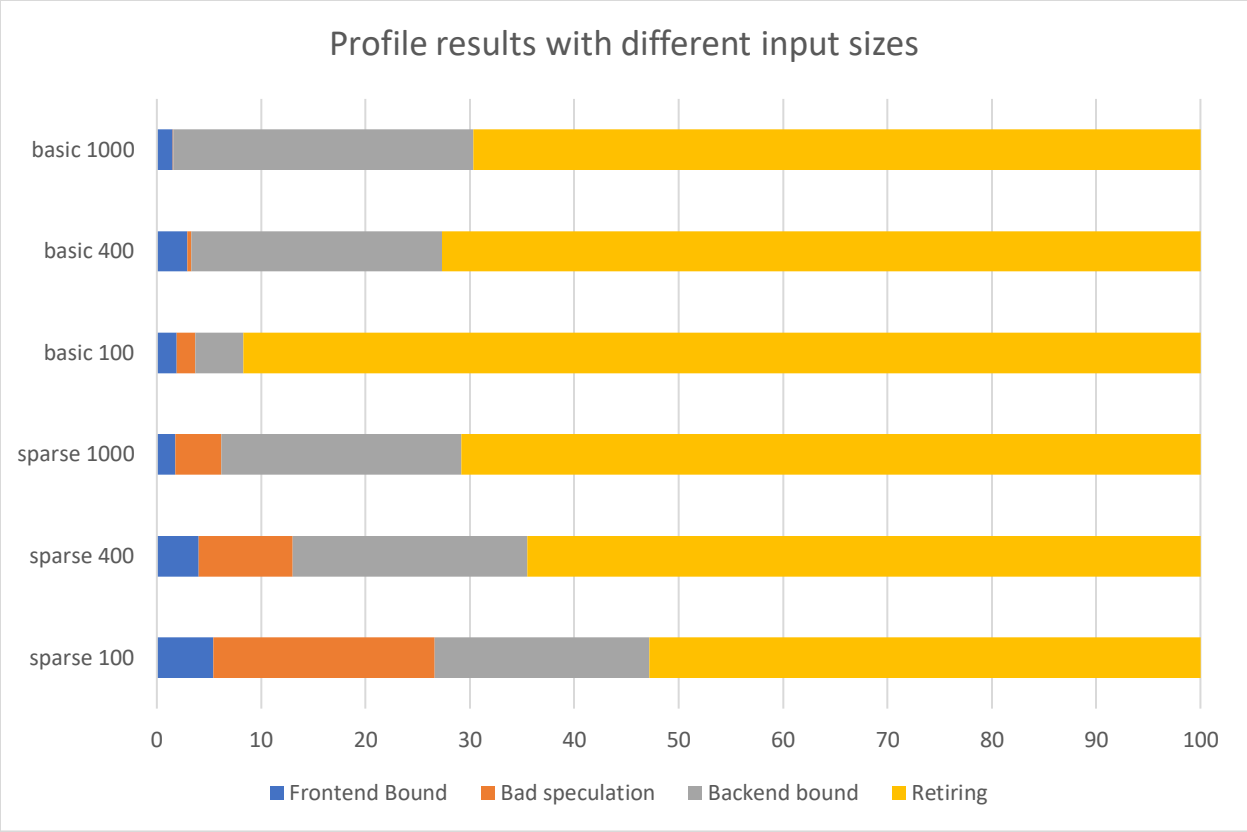
Run time analysis (in seconds)

Input size / Matmul	100	400	1000
Basic	0.00636	0.366	7.57
CSR	0.000706	0.0257	0.349

All in all, as input size increases, for sparse matrix multiplication, matmul_sparse performs better at least by 10 times in terms of runtime.
To generate sparse matrix, I wrote this function such that roughly 80% of the time, the pseudo random number returned is 0. It fills up each input matrix with such numbers.

```
int limited_rand() {
    int res = rand() % 100;
    if (res < 80) {
        res = 0;
    }
    return res;
}
```

I recorded the runtime of each multiplication 10 times and take the average, for both profiling and recording runtime. Except for input size 100 which is small enough for the process not to timeout and cut halfway, I ran the multiplication for 100 times so that it dominates the runtime for profiling purposes. For recording runtime, a timer that starts right before the line to call the function matmul_sparse however many times, and ends right after, is used and the time elapsed is printed onto the console.



Front end (FE) bound differences:
CSR has more conditional statements, due to the need to check for various conditions while looping over the 3 arrays in CSR format. This evidently introduced more front-end stalls due to branch mispredictions.
Backend (BE) bound differences:
As CSR focuses on non-zero elements, it can be more cache-friendly, leading to reduced memory latency.
Bad speculation differences:
Given the more complex control flow in, there's a higher chance of mispredictions that lead to bad speculation. The CPU might waste cycles on wrongly speculated branches before backtracking. In terms of percentage, evidently, the portion of speculation in matmul_sparse is significantly manifolds higher.
Retiring:
The rate of retiring instructions might vary. While skipping zero elements can speed up computation, the conditional statements and indirect addressing can occasionally introduce stalls.

Debugging
Looping over the 3 arrays was a little complicated due to their interwoven logic, and also the first matrix had to be considered row first while the second, column first, so it was a little confusing to keep track of the loops. All in all, I managed to get it to work correctly by writing on paper to keep track of which loop goes from where to where and which elements to be multiplied to which sum.