**Database System Implementation** **Project 2**
COMS 4112 Harish Karthikeyan - HK 2854
Prof. Kenneth Ross Thiyagesh Viswanathan - TV 2219

## Problem Statement

1. To create a pointer-free tree structure for in-memory search trees. Each level of the tree is associated with a separate array containing the keys at that level.
2. To probe based on binary search at each node to ultimately return the leaf page to which the probe key belongs.
3. To probe based on Intel SSE instruction set for specific fanouts of 9,5 and 17
4. To hardcode a probing mechanism for a specific instance of 9-5-9 trees using the SSE instructions set with other optimizations.
5. To compare the performance of each of the probing methods and report the findings.

## Investigation/Research

### Creation of pointer free structures

The pointer free tree structure is similar to a B-Tree but with the tree construction happening bottom up (bulk loading). We maintain an array for each level and store every key belonging to one particular level to the array. We use `posix_memalign` to align each array at 16 byte boundaries. This maps the nodes to their respective children without usage of pointers. The tree was constructed from a sorted set of randomly generated 32-bit integer keys. The set is of size **K** where **K** is provided by the invocation call:

```
./build K P <Space-Separated Fanout Values from the Root>
```

**P** determines the number of probing keys to be generated at random which is to be used in the next section.

### Probing-Binary search

The probing is based set of randomly generated 32 bit integer keys, the number of which depends on user's input. For each probe key that is generated, we recursively perform a binary search at a node in each level. The results from binary search of one level helps to locate the node in the next level. Now binary search is performed in this level. This process is repeated until the resulting value from leaf level is returned.

### Probing-SSE instruction

In this method, we rely on Intel's SSE instruction set instead of performing a binary search. This method is adapted from the work of Polychroniou and Ross [PR14]. For trees with fanouts of 9, 5 and 17 we load delimiters of a node in that level into SIMD register of 128 bits. This register is of type `__m128i`. The probe key is broadcast to all lanes of another SIMD register. We do a SIMD comparison using the `__mm_cmplt_epi32(x,y)` and convert the result to a bit mask. The result of this bitmask determines the node in the next level and the process is repeated until all levels are exhausted. The result of the leaf level is then returned.

### Probing-Hardcoded

In this method, we use Intel SSE instructions set to probe on a specific tree of size 9-5-9. The entire loop across the three levels is unrolled with additional optimization of probing 4 keys at-a-time. This essentially means that level 1 access is done for all the four keys and then level two is done for all the four and so on. In addition the root node of the index is explicitly loaded into the register variables `lvl_0_a` and `lvl_0_b` at the beginning of the entire probe so that it is not reloaded at every probe phase. This harcoded probe function is effectively without any looping and conditional constructs.

## Machine Configuration

The code was executed on the CLIC machines remotely through an SSH tunnel. The system particulars are

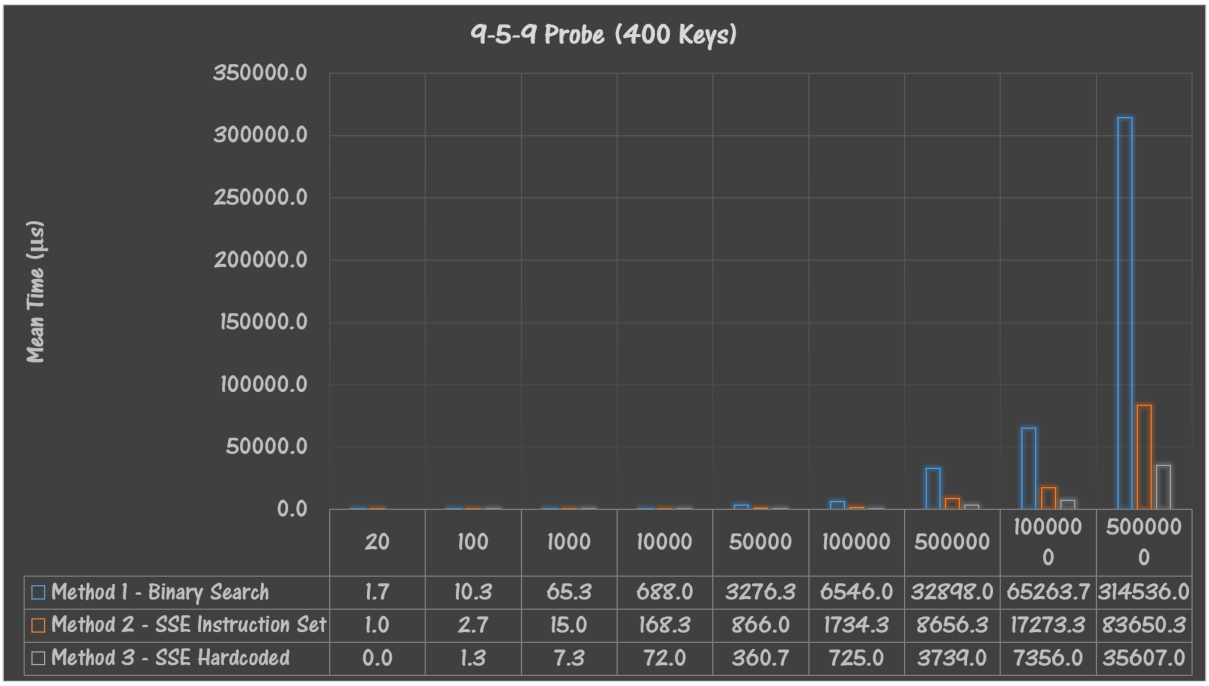
- Operating System: Ubuntu Linux 12.04 (64 bit)
- Processor: Intel Zeon CPU X5650 2.67 GHz
- RAM size: 8GB
- Compiler: GCC using GNU 99 standard of C
- The program was compiled using the following optimization flags

  ```
  -O3 -flto -march=native
  ```

- In addition, `-pedantic` flag was used as an option for gcc.
- The system was idle apart from the experiment.
- `<sys/time.h>` package was used to provide the `timeval` structure to measure the start and end time in microseconds.

## Experimentation

### 9-5-9 Trees

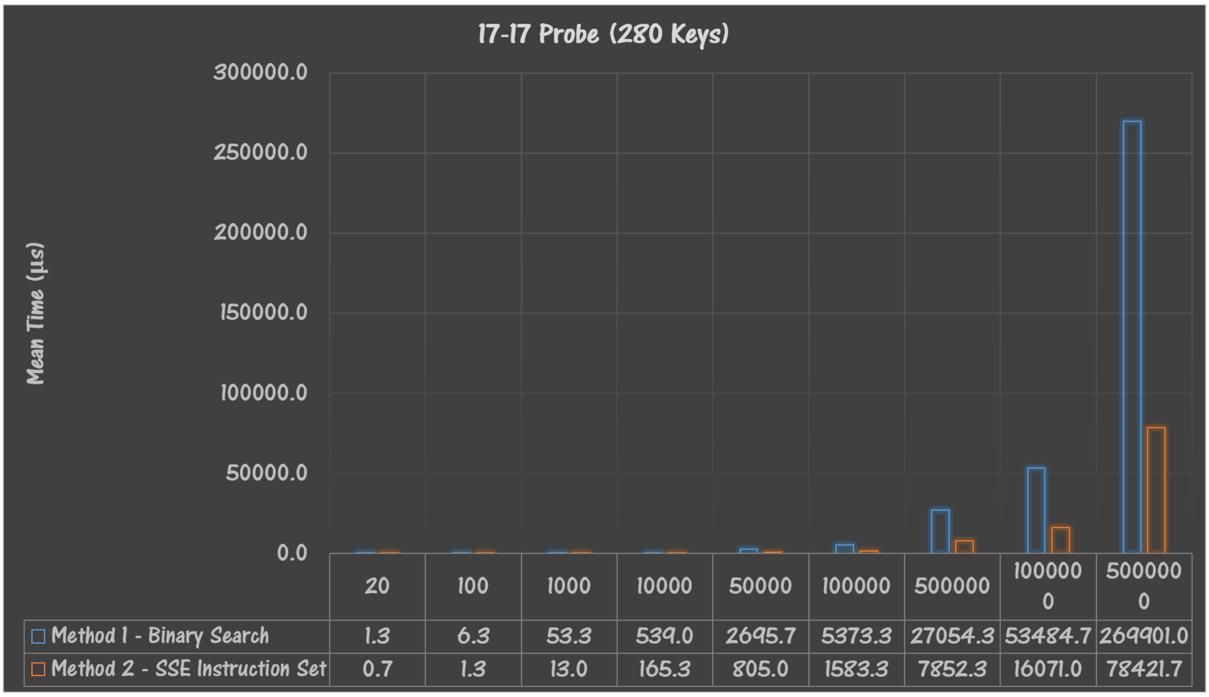


| | 20 | 100 | 1000 | 10000 | 50000 | 100000 | 500000 | 1000000 | 5000000 |
|---|---|---|---|---|---|---|---|---|---|
| □ Method 1 - Binary Search | 1.7 | 10.3 | 65.3 | 688.0 | 3276.3 | 6546.0 | 32898.0 | 65263.7 | 314536.0 |
| □ Method 2 - SSE Instruction Set | 1.0 | 2.7 | 15.0 | 168.3 | 866.0 | 1734.3 | 8656.3 | 17273.3 | 83650.3 |
| □ Method 3 - SSE Hardcoded | 0.0 | 1.3 | 7.3 | 72.0 | 360.7 | 725.0 | 3739.0 | 7356.0 | 35607.0 |

Figure 1: 9-5-9 Probing

For 9-5-9 tree with 400 randomly generated keys, probing was performed using the three methods for varying number of probe keys. The probing was done on randomly generated key sets of size 20, 100, 1000, 10000, 50000, 100000, 500000, 1000000 and 5000000. The timing was measured and plotted as a graph as shown in Figure 1. For better accuracy the experiment was done three times for each key set size and the average plotted, rounded up to 1 decimal place. From the graph it is evident that for the same set of probes the time taken to probe in a hardcoded version is less than half of the time taken to probe in a non-hardcoded version. Binary search version lags much behind.

## 17-17 Trees

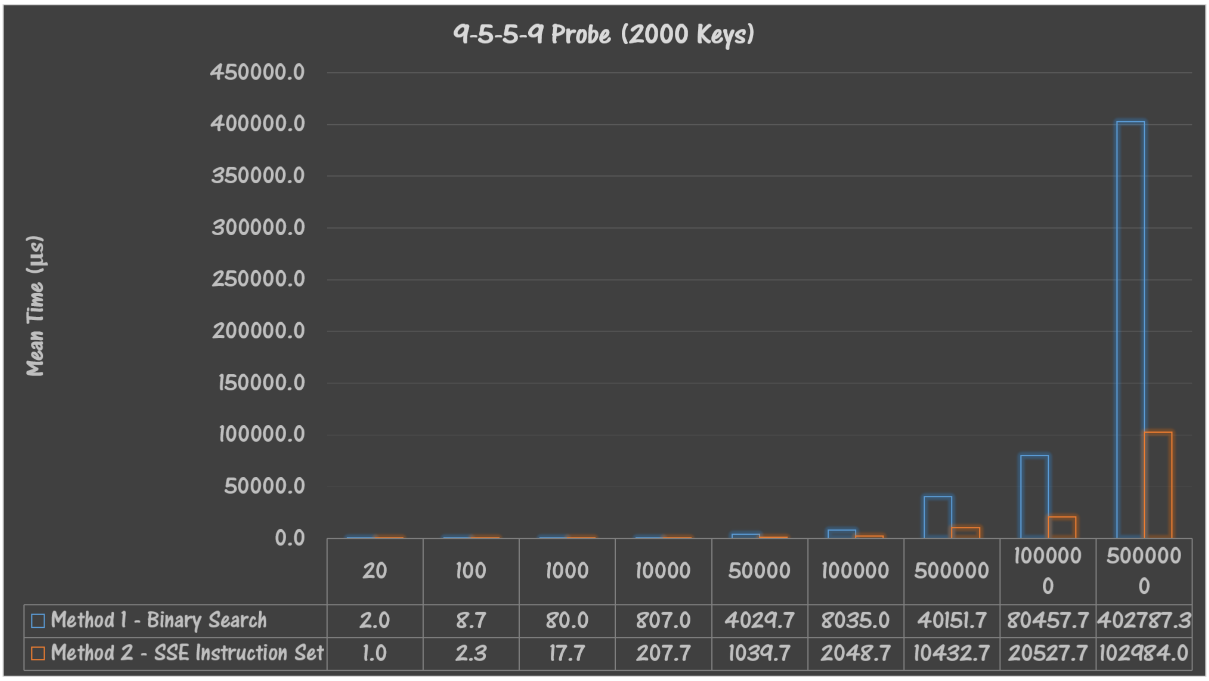


| | 20 | 100 | 1000 | 10000 | 50000 | 100000 | 500000 | 1000000 | 5000000 |
|---|---|---|---|---|---|---|---|---|---|
| □ Method 1 - Binary Search | 1.3 | 6.3 | 53.3 | 539.0 | 2695.7 | 5373.3 | 27054.3 | 53484.7 | 269901.0 |
| □ Method 2 - SSE Instruction Set | 0.7 | 1.3 | 13.0 | 165.3 | 805.0 | 1583.3 | 7852.3 | 16071.0 | 78421.7 |

Figure 2: 17-17 Probing

For 17-17 tree with 280 randomly generated keys, probing was performed using the two methods for varying number of probe keys. The probing was done on randomly generated key sets of size 20, 100, 1000, 10000, 50000, 100000, 500000, 1000000 and 5000000. The timing was measured and plotted as a graph as shown in Figure 2. For better accuracy the experiment was done three times for each key set size and the average plotted, rounded up to 1 decimal place. We can observe that for the same set of probes the time taken to probe using the SSE instruction set performs better.

**9-5-5-9 Trees**



| | 20 | 100 | 1000 | 10000 | 50000 | 100000 | 500000 | 1000000 | 5000000 |
|---|---|---|---|---|---|---|---|---|---|
| □ Method 1 - Binary Search | 2.0 | 8.7 | 80.0 | 807.0 | 4029.7 | 8035.0 | 40151.7 | 80457.7 | 402787.3 |
| □ Method 2 - SSE Instruction Set | 1.0 | 2.3 | 17.7 | 207.7 | 1039.7 | 2048.7 | 10432.7 | 20527.7 | 102984.0 |

Figure 3: 9-5-5-9 Probing

For 9-5-5-9 tree with 2000 randomly generated keys, probing was performed using the two methods for varying number of probe keys. The probing was done on randomly generated key sets of size 20, 100, 1000, 10000, 50000, 100000, 500000, 1000000 and 5000000. The timing was measured and plotted as a graph as shown in Figure 3. For better accuracy the experiment was done three times for each key set size and the average plotted, rounded up to 1 decimal place. We can observe that for the same set of probes the time taken to probe using the SSE instruction set performs better.

## Inference

It is observed that for all the three tree structures and for all the key counts the binary search takes the longest toe complete probing. This is for the simple reason that at a node, binary search keeps making repeated calls owing to its recursive nature. It is observed that for all the three tree structures and for all the key counts the binary search takes the longest toe complete probing. This is for the simple reason that at a node, binary search keeps making repeated calls owing to its recursive nature. This increases the complexity. In the SSE Instruction set, for a single node the whole processing happens in constant time. This makes the SSE Instruction set far more efficient when compared to the recursive Binary Search Algorithm. The hardcoded probing function is even more efficient for the simple reason that there is no loop, not even to make an access at each level. The entire loop is unrolled and this reduces the time taken. Also the hardcoded version does the access 4 at a time and this makes it more efficient. The SSE based probing overcomes this latency by trying to accelerate the computation of finding the right node to search for in each tree level using SIMD registers. The harcoded has the best performance of the lot as it is able to probe up to four keys at a time and the the root node of the index is explicitly loaded into the register variables at the beginning of the entire probe cutting down the time spent in reloading at every probe phase. Leveraging the SSE Instruction Set is a prudent choice that makes significant gains in the search and the difference grows more pronounced as the number of probe keys increases.

## Attachments

- Readme.md
- Makefile
- Test files
- main.c
- tree.c
- tree.h
- p2random.c
- p2random.h

## References

[PR14] Orestis Polychroniou and Kenneth A. Ross. A comprehensive study of main-memory partitioning and its application to large-scale comparison- and radix-sort. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 755–766, New York, NY, USA, 2014. ACM.