CSE5242: Advanced DBMS

Dr. John Paparrizos

Fall 2023

December 3rd, 2023

Project2

Chiluvuri Hrushik, Jiyong Kwag, Nirmal Philipose Mathew

Abstract

Search algorithms are utilized on many databases management systems to efficiently look up specific single item or range of item. The simplest method of search algorithm is sequential search. Sequential search traverse through the database from one end to another to find item. Sequential search is not only easy to implement but also easy to understand its concept. However, sequential search suffers from cost of O(N) running time in worst case scenario where algorithm needs to search to end of the database. In our project, we leveraged binary search. Binary search is one of the types of search algorithm where size of array repeatedly halved to find specific item. Binary search algorithms achieve O(log N) running time in worst case scenario, which is a significant improvement from the O(N) runtime that results from sequential search. Furthermore, we take one step further on implementing binary search where we used different types of binary search to compare efficiency. The first method is control dependent and data dependent binary search. The second method is data dependent parallel binary search to find multiple targets in the array and binary search with Single Instruction Multiple Data (SIMD) using AVX2 to find multiple targets in the array. Lastly, our project implemented band join. Band join is type of inner join where algorithm looks for range of item from outer array to inner array. We used data dependent parallel binary search and binary search with SIMD to implement band join and compared their performances based on running time with different parameters such as size of inner array, outer array, and range. Thus, through comparison of these three methodologies, our project 2 aims to analyze the effectiveness of data and control dependent operations, SIMD, and band join.

Control Dependency vs. Data Dependency

According to the article Advances in Computers, statement t is control dependent of statement ‘s’ if statement ‘s’ is a conditional statement and execution of statement ’t’ depends on the result of statement s [1]. In simplicity, we can think of control dependency as if-else statement in programming languages. For data dependence, statement ‘t’ is data dependent on statement ‘s’ if statement t defines new variable ‘v’ and variable ‘v’ affect the result of statement ‘s’ [1]. In our case, both control dependence and data dependence are similar in the way that statement ‘t’ is affected by statement ‘s’. However, control dependency directly affects the control flow of the system while data dependency affects the statement without manipulating the control flow of the system. In our project, both data and control dependent binary searches are implemented to compare their efficiency when running the algorithms on datasets of varying size.

```
while(left<right) {
  mid = (left + right)/2;
  if (data[mid]>=target)
    right=mid;
  else
    left=mid+1;
}
```

```
//0^1 = 1 and 1^1 = 0
while(left<right) {
  mid = (left + right) / 2; //get middle
  int64_t flag = data[mid] >= target; //data[mid] >= target ? 1 : 0
  right = flag * mid + (flag^1) * right;// data[mid] >= target ? mid : right
  left = flag * left + (flag^1) * (mid+1);// data[mid] >= target ? left : mid+1
}
return right;
```

Fig. 1. Control Dependency (left) and Data Dependency (right)

Figure 1 directly comes from the project 2 implementations of binary search. From figure 1, the initialization execution of right and left variables is affected by the control statement if-else statement. For the data dependency, right and left variables are executed without changing the control flow of the program. However, the initialization of the right and left variable is data dependent on the flag variable’s comparison execution.

SIMD

According to High Performance Parallelism Pearls, SIMD boosts the computation speed to reduce the running time of the algorithm by leveraging the hardware component with executing single instruction

on multiple data at same time [2]. When running multiple data, data are stored in vectors to represent the data. Then, single instructions execute the commands, which include arithmetic and comparison operations, on the vector in parallel to generate the result.

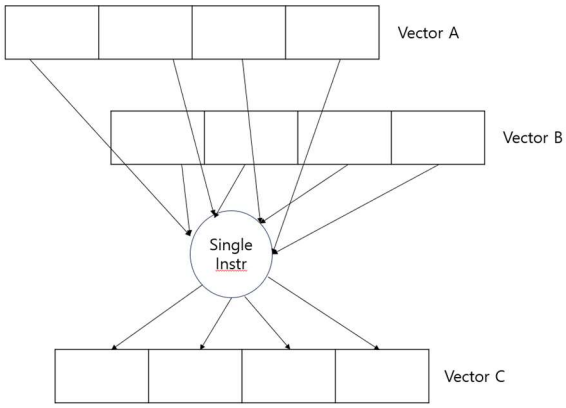

Fig. 2. Single Instruction running multiple data [2].

From figure 2, two vectors, vector A and B, are executed with single instruction and result in one vector that contains the result of the instruction. In project 2, within the OSU stdlinux server, we performed the SIMD functions based on Intel AVX2.

Band Join

According to Oracle Tunning Guide, join is referred to as combining multiple tables' rows into one to easily interpret data that are spread out across the table [3]. By combining the multiple tables into one, it helps users to easily understand and find appropriate statistics or analysis on the data. There are multiple types of joins: inner join, outer join, and semi join. Each join has its own functionality for various cases. In project 2, we mainly focus on inner join, especially band join. Inner join is "a join that returns only rows that satisfy the join condition [3]." Similarly, band join takes key value from the outer array and searches for the matching range of data in the inner array. In our project 2, we aim to implement band join through parallel binary search and SIMD binary search to compare the performance of parallel versus linear computation.

CPU and Cache

Central processing unit (CPU), cache, main memory, and disk play an essential role when a computer needs to execute programs with a lot of data and complex algorithms. Each system has a unique role when they are storing data and execute the algorithm. CPU is a hardware component that executes the machine's operating system and performs mathematical operation [5]. However, CPU cannot store data and utilizes other hardware to store the data. Thus, various types of hardware are utilized to store the data for different purposes. Cache and main memory are high speed data storage that is close to CPU [6, 7]. They are mainly used for storing recent or temporary data for CPU computational calculations. Cache stores less data than main memory but has faster accessibility to CPU than main memory. More memory size for cache and main memory means the system can store more temporary data for CPU computational execution. However, both cache and main memory suffer from size of their storage and volatility when the machine is turned off. Thus, disk is used to store data permanently. However, access time for disk suffers latency due to its size and lower accessibility from the CPU. Thus, the machine utilizes three kinds of data in order to process data execution and storage. Figure 3 compares the size and speed of each hardware.

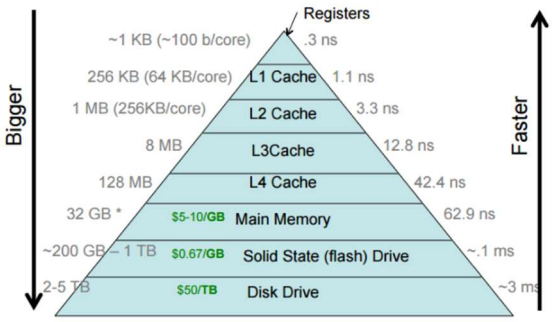

Fig. 3. Memory hierarch with memory size and speed [4]

In project 2, we are analyzing how the CPU and cache affect the result. Since CPU is the brain of the machine, CPU acts as core hardware for calculating mathematical operation and algorithm in the system. Specifically, CPU speed is determined by the number of calculations that the CPU makes per cycle, and this is called clock speed [8]. Clock speed is represented as GHz. For example, 1 Hz can generate 1 digital signal for instruction execution. A higher GHz means the CPU runs more operations than a lower GHz CPU. We also must consider the number of cores and threads for calculation time. It is essential to have more cores and threads to execute the program using parallel processing. Thus, the more cores and threads the system has, the greater the number of calculations that can be completed.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2670 v3 @ 2.30GHz
Stepping:              2
CPU MHz:               2297.340
BogoMIPS:              4594.68
Hypervisor vendor:     Microsoft
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-3
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx lm constant_tsc rep_good nopl eagerfpu pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_
2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm invpcid_single ssbd rsb_ctxsw ibrs ibpb stibp fsgsbase bmi1 avx2 smep bmi2 erms invpcid xsaveopt spec_ctrl intel_stibp flush_l1d
```

Fig. 4. OSU stdlinux information.

Figure 4 shows the OSU stdlinux system by lscpu command. The OSU stdlinux has model Intel Xeon CPU E5-2670 2.30 GHz. As we explained previously, 2.30 GHz represents the clock speed of the system. Thus, 2.30 GHz CPU can generate 23 billion digital signals per second. The system has one socket, which represents the physical number of CPU that the system has and has 4 cores per socket, which means the system has a total of 4 cores in our case. Also, since each core has one thread and the system has a total of 4 cores, the number of logical cores is 4 (which shows as CPU(s) in forth row of the figure 4). Lastly, the system has L1d and L1i cache memory size as 32 KiB and L2 cache with 256K. Through CPU, core, and cache analysis, higher frequency or clock speed CPU has the system can calculate more data per second. For number of cores and threads, the more cores and threads we have, the system can perform more concurrent executions. Then, lastly, if the system has higher memory for cache, the system can hold more temporary data which enables CPU to compute more data per execution.

Profiling the Binary Search

In project2, implementations of binary search are divided into four parts:

1. Binary search with control dependency.
2. Binary search with data dependency

A. With arithmetic operation

B. Without multiplication operation

3. 4x Binary search with data dependency without multiplication
4. 4x Binary search with SIMD using AVX2.

Our primary goal is to compare running time among different implementations of binary search. To profile the binary search with different size of N, we used repeat parameter of R = 1000 to test stable result of binary.

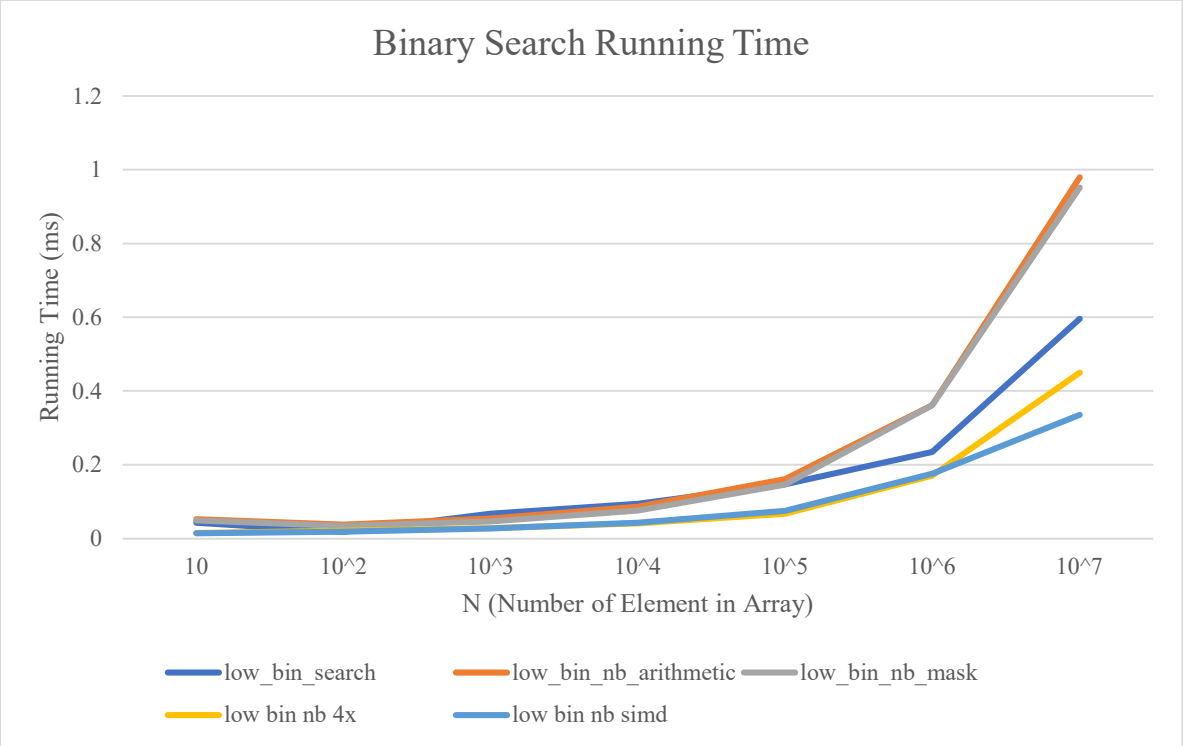

Fig. 5. Binary Search Running Time Comparison

From figure 5, we can see that running time of all five binary search implementation increases exponentially as number of elements in array (N) increases by 10. The running time of all different binary searches is similar until N is 100 as they are all quick at low volumes. Then, running time begins to increase as N increases. Especially when N is 10^7, we can see distinct differences among different binary searches. Binary search with SIMD shows the fastest running time among different binary search implementations, and 4x binary search and binary search control dependency are second and show similar running time, while 4x binary search outperform binary search with control dependency. Then, the binary search with data dependency (with arithmetic operation and without multiplication operation) is slower and shows similar running time compared to other binary search operation. However, the binary search with nb_mask (data dependency without multiplication) shows slightly better performance than arithmetic binary search. Thus, by comparing the graphed results, we can conclude several results:

1. Different data dependency implementations (with and without arithmetic operation) do not significantly affect the running time of the binary search.
2. When searching for one value in the array, control dependency shows better performance than data dependency.

3. However, when we allow concurrent searches, data dependent binary searches outperform control dependent binary searches.
4. Lastly, binary search with SIMD using AVX2 outperforms all other binary search variants.

Profiling the Band Join

Case 1: Increasing N, X and Y with same Z(bound)

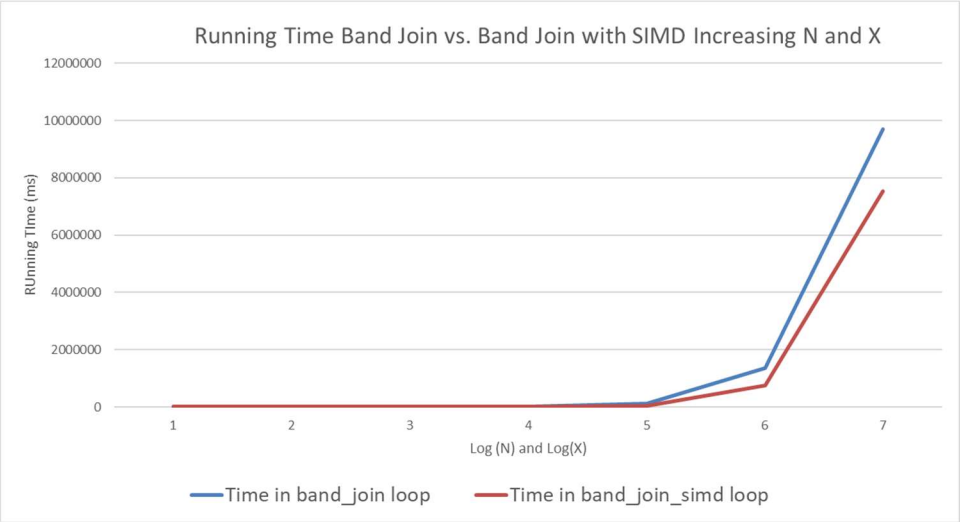

Fig. 6: Time for joining when N and X are increasing exponentially.

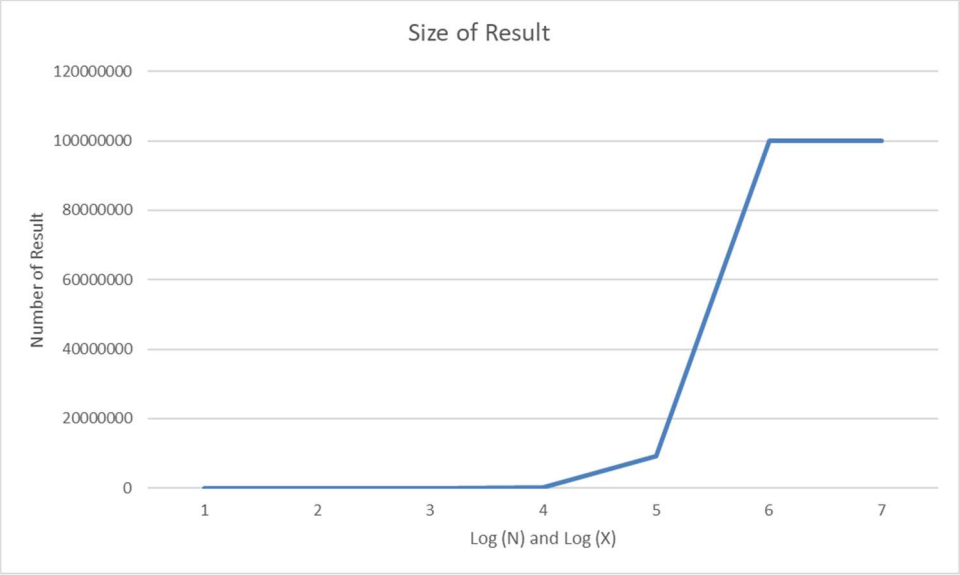

Fig. 6.1: Size of result array is increasing exponentially.

Figure 6 illustrates changes in running time as size of inner array (N) and size and size of outer array (X) are increasing by factor of 10 until 10^7 while having same bound (Z) as 10^6 and size of result (Y) as 10^8. As both N and X increase, running time of both band join and band join with SIMD exponentially increase. Moreover, when both N and X are below 10^6, running time of both band join and band join with SIMD do not show significant differences. However, after size of N and X reach to 10^6, running time of both band join and band join with SIMD gradually increases. Even though running time of both band join and band join with SIMD continuously increase, band join with SIMD shows slight better performance than simple band join. When we compare the running time difference

between band join and band join at 10^6 and 10^7, we can notice that performance difference begins to have more gap between them. Thus, we can conclude that if we keep increasing the number of inner array (N) and number of outer array (X), running time of both band join and band join with SIMD continuously increases. However, band join with SIMD shows significantly better performance as both N and X increases.

Case 2: Increasing Z value (N and X smaller values)

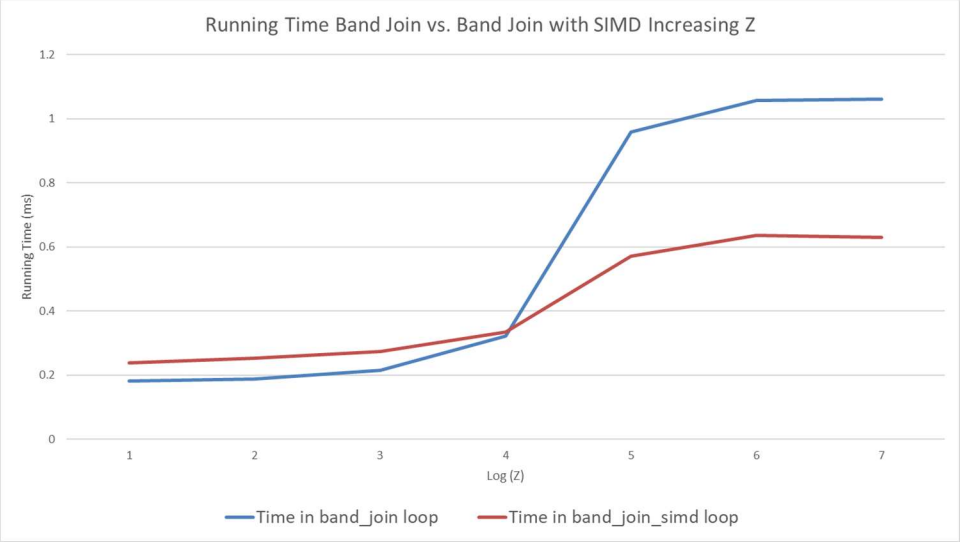

Fig. 7. Band join running time with increasing Z value

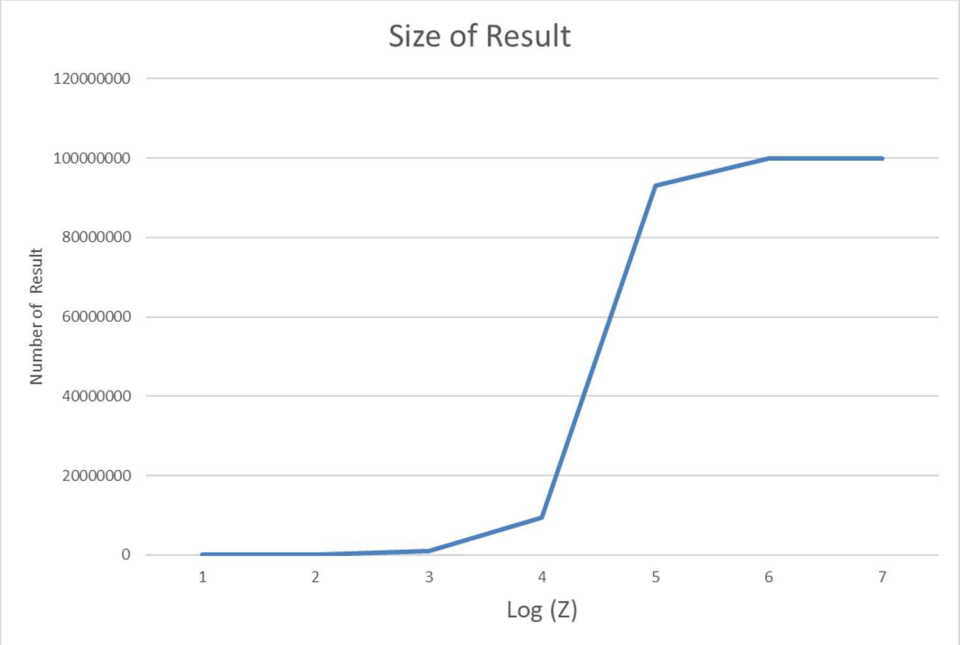

Fig. 7. Band join size result with increasing Z value

From case 1, when N and X are 10^6, the running time of both band join and band join with SIMD begin to show differences. Thus, in case 2, we used N and X as 10^6 to compare performance differences when the bound (Z) value is changing. Input Z represents the range of the outer array in

band join. When we increase the Z value, the band join searches for more data in the inner array. From figure 7, as Z value increases by factor of 10 until Z is 10^7, number of result data exponentially increase. As a result, the running time of both band joins with 4x data dependent binary search and band join SIMD exponential increases. However, at the beginning when Z value is between 10 and 10^3, performance of band join is slightly better than band join with SIMD. However, as Z value passes 10^4, running time of band join significantly increased while the running time of band join with SIMD slightly increased. This is caused by the parallel nature of SIMD which offers better performance than band join with simple binary search.

Special Note

During debugging phases, our team tested the greater than operator as opposed to greater than or equal to within the binary search algorithm when determining whether the target value is on the left or right side of the middle index. We performed these experiments because the OSU stdlinux couldn't compile "`_mm256_cmpge_epi64_mask`" operation with inlining failure error, thus we used only "`_mm256_cmpgt_epi64`" to test band join with SIMD (Later, we combined greater than and equal to operation using or operation to generate greater than or equal to operation). For binary search with greater than, it does successfully calculate indices of targets values like other binary searches with greater than or equal to. However, when combining it with band join and extending bound by 1, band join with greater than gives slightly different result from binary join with greater than or equal to and upon testing, we believe these results were logically correct although the coding logic we used was incorrect, so we are sharing this information.

654729	906624		654729	906624
809590	906900		809590	906900
679675	906971		679675	906971
492049	907061		492049	907061
			72085	907206

Fig. 8. (Right) Band Join Result with Greater than or equal to and (Left) Band join with Greater than.

155145568	155145560

Fig. 9. Inner array value and outer array value of index 72085 and 907206

From figure 8 and 9, we used command "db5242 1000000 1000000 100000000 10 1". Band join with greater than or equal to (GE) and band join with greater than (GT) provide same result until index (492049, 907061). However, band join with GT provide more results which is (72085, 907206). The value of the index (72085, 907206) is (155145568, 155145560). Since our bound in 10, this index should be included. However, in fact only greater than operator in binary search is able to find it. As our group talked through and performed some test cases, greater than sign cannot achieve binary search correctly, so we stuck with the greater than or equal to comparison. However, we still came up with interesting outputs as a consequence. Thus, we included this explanation in the report.

Conclusion

Our project 2 focused on simple algorithms to search and perform joins on large datasets. While the primary algorithm used was binary search, we leveraged the power of parallel computing to perform significantly efficient computation for similar operations. We learnt how control dependency influences system flow and how data dependency impacts statements without changing system flow. In practical situations, control dependency exhibited enhanced performance for small datasets, and data dependency works efficiently for larger datasets and finding multiple targets in parallel. We used SIMD(Single Instruction Multiple Data) using AVX2 in stdlinux for both binary search and join function. We noticed significant performance boost when compared to normal functions that we developed. The profiling on

binary search and join function directly showed the potential of using SIMD in computational processing.

This project helped us in understanding the implications for efficient data processing, specifically for the field where data retrieval and data analysis are essential. Future learning explores the application of these findings in real-world scenarios like big data analytics, database management where efficiency is paramount. As data driven computational needs continue to evolve, these insights will help us develop efficient data processing methods.

Contribution

Chiluvuri Hrushik

- Code Implementation
- Profiling and Analysis
- Writing Report

Jiyong Kwag

- Code Implementation
- Profiling and Analysis
- Writing Report

Nirmal Mathew

- Code Implementation
- Profiling and Analysis
- Writing Report

References

1. Zelkowitz, Marvin V. Advances in Computers. Academic Elsevier, 2008.
2. Jeffers, Jim. High Performance Parallelism Pearls. Morgan Kaufmann.
3. ″SQL Tuning Guide.” Oracle Help Center, 20 Dec. 2021, docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/joins.html#GUID-BD96F1B4-76D4-43DF-98B6-D07F46838C4A.
4. ″Lab 4: Caching.” Lab 4: Caching - HackMD, cs.brown.edu/courses/csci1310/2020/assign/labs/lab4.html. Accessed 20 Nov. 2023.
5. ″What Is ...” Amazon, The University, 1978, aws.amazon.com/what-is/cpu/.
6. What Is Caching and How It Works | AWS, aws.amazon.com/caching/. Accessed 22 Nov. 2023.
7. Org, Kjell At Ieee Dot. Main Memory, chortle.ccsu.edu/java5/notes/chap01/ch01_6.html. Accessed 22 Nov. 2023.
8. Clock speed is one of your CPU’s key specifications — but what does it really mean? Clock speed is one of your CPU’s key specifications — but what does it really mean? “CPU Speed: What Is CPU Clock Speed?” Intel, www.intel.com/content/www/us/en/gaming/resources/cpu-clock-speed.html. Accessed 22 Nov. 2023.