How Modern Processing Devices are Being Adapted to optimize Database Operations

Saloni Verma
University of Magdeburg, Germany
saloni.verma@st.ovgu.de

ABSTRACT

With the popularity of memory-based database systems comes the need for optimized, accurate and swift query processing. Adapting the database to perform optimally using co-processors, and GPUs is fast becoming a commonplace practice. This type of hardware tuning does not always translate well on other hardware devices, primarily because of basic design differences. For the optimization of database operations different software engineering techniques developed recently would be delineated. These techniques might not necessarily scale well for different database systems. In this paper, we narrate the state of the art for implementing and optimizing database perations using modern hardware devices. We survey the different approaches of maximizing database performance by using FPGAs, GPUs, Vector Processing Units, etc, and compare their scalability and shortcomings. We make a point for amalgamating hardware tuning and database-algorithms for comprehensive database optimization. We also mark out the advancements in network distributed database management, ideal for large databases. We mention the current advancements in using GPUs for databases and techniques for managing infinite data-stream online systems.

1 INTRODUCTION

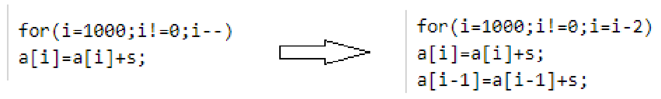
As businesses and networks grow, the simultaneous growth of data has been exponential. Managing such large databases, and ensuring a quick and efficient system is also becoming highly specialised. Numerous methods for query processing and its optimization have been developed catering to the needs of the environment, and the type of processing device(s) available. Various techniques involving hardware tuning have been developed depending on the requirement, but these methods do not perform as well when the underlying hardware is changed. For example, GPUs are a preferred method of performing complex parallel processes which in turn leaves the CPU free for other tasks. Since GPUs come with multiple cores, they deliver high performance and efficient query processing as compared to regular processors. There has been incredible growth in GPU development and they are a good, cheap method of accelerating database performance.

Similarly, other methods of hardware tuning have also been developed which can be adapted for use in database management systems. Each method has its advantages and short-comings which we will outline here. Therefore, finding the optimal method for a particular database is an intensive process, which is why extensive research exists trying to find novel solutions to this crucial question. Another very important fact is that software engineering along with hardware tweaks can result in efficient, scalable optimization methods [5].

There is also a rising trend of having distributed storage databases - either across a network or on multiple computers. Since they do not share a common processor, the processing solutions for such systems differ from traditional databases. The transactions on such systems can be processed by multiple processors which may improve performance[3]. It is also imperative to see that such systems offer an additional advantage of security and these transactions should be efficient, since not all data is stored at one place.

1.1 Background

According to Moore's Law, we can improve the performance of any database by upgrading its hardware, scale by adding more power like CPU, RAM to an existing machine. This process is called Vertical Scaling [26]. However, in Horizontal scaling we can scale by adding more machines into the total pool of resources. We know that CPU access times have not increased much, which means CPU utilisation continues to remain low. Since the processing power of CPUs has increased, but the data access rates have not improved that well, the processor remains idle in the time that it takes for data fetch. This is the *Memory Wall* limitation. Moreover, more power would be required to improve clock rates, which would lead to more heating issues. This is the *Power Wall* limitation [20]. And due to these issues, we have to rely on heterogeneous systems - CPU along with other co-processors to yield results quickly. Since the size of databases has grown manifold, CPUs alone cannot take the brunt of processing. Which is why modern systems have had to rely on alternate processors, and respective optimization technique to find the right balance. Multiple vendors specialise for one particular processor, but finding one solution that works across the



```
for(i=1000;i!=0;i--)
a[i]=a[i]+s;
```

```
for(i=1000;i!=0;i=i-2)
a[i]=a[i]+s;
a[i-1]=a[i-1]+s;
```

Figure 1: Simple Example of Loop Unrolling

board is difficult because those algorithms fare less on other processors, for which they were not specifically designed.

Join Order optimization is the process of determining the order in which tables would be joined for producing the result of a query. A join order algorithm defines the order of the tables. It is important to avoid the invalid pairs like joining a table to its own again. There are various methods of finding out the optimal join order including Deterministic and Randomized. *Deterministic approaches* consist of the Greedy and Exhaustive search. In the Greedy algorithm, the cheapest branch is opted for in each step. They have a short run time and use simple heuristics. They do not guarantee efficiency or optimal solution but a quick solution is guaranteed. In the Exhaustive method, small solutions are found in each iteration to completion. It finds out an optimal solution, can run for long times and are best suited for simple queries. There are also *Randomised approaches* for determining the Join Order which includes Transformation, Sampling, and Genetic Algorithms, etc. On a given input, different algorithms would produce unique outputs, which may be efficient but not necessarily optimal. In Transformation, an initial result is selected, and the current result is transformed. In Sampling, some candidates for joining are randomly chosen and the best solution is stored. In Genetic algorithms, there occurs a selection from an initial pool, and new solutions are created based on the solution pool. A third option is *Hybrid approach* which combines the advantages of both deterministic and randomised approaches and is suitable for complex optimization problems.

The Main memory databases are optimized by hardware tuning to process all operations. Vectorisation (Single Instruction Multiple Data - SIMD), and loop unrolling are some code-optimization methods, which in conjunction with processors have shown better performance in some cases. In main memory databases, the new bottleneck occurs between the CPU and data, while in disk-based databases it used to between the Disk and . To break the memory wall and better utilise the CPU, better methods of cache awareness were developed. There are two alternative processing models in databases. The Operator-at-a-Time bulk processing model of query execution and the Tuple at a time model Volcano Model.

In **Tuple-at-a-Time Execution**[15], the operator requests tuples from input using *next()*, processes it, and passes it on to the next operator. It creates small intermediate results that can be stored in the main memory easily. There is pipeline parallelism. Operators are interleaved, which can result in data and instruction cache misses. The Instruction per cycle (IPC) ratio is low, and much time i spent on field access and less than ten percent of the time is spent on actual query execution. It is hard to implement vector (SIMD) instructions since there are polymorphic operations. Alternatively, in **Operator-at-a-time processing**[21], each operator processes the whole input and outputs the entire result. Each intermediate result is materialised in the memory. There is no particular pipelining, and each operator runs just once. Function calls are replaced by tight loops, that fit into instruction caches. We can also utilise CPU features like hardware prefetching. This can be optimized using loop unrolling and vectorisation. *Loop Unrolling* is a transformation technique used for optimizing a program's execution by removing iterations. It is at the expense of the program's size and can be done manually or by an optimizing compiler as shown in Fig 1.

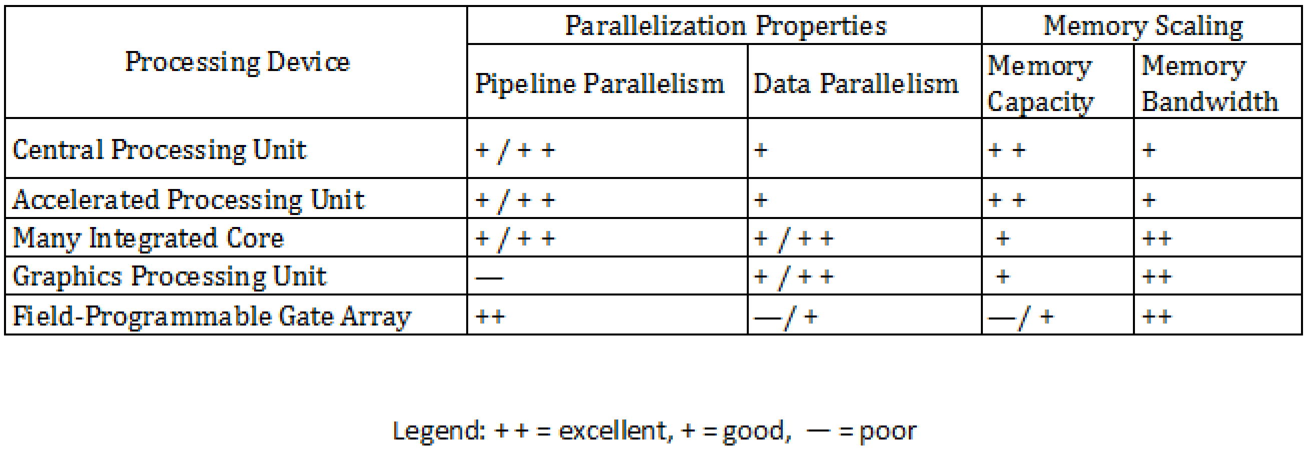
The **Vectorised Execution model** combines features of both the Operator-at-a-time and Tuple-at-a-time processing. The iteration is volcano style but at each *next()* call, a large number of tuples are returned for processing which is called a 'vector'. The vector size should be chosen in a manner that it is large enough to cover the function call overhead and small enough that it does not thrash data cache. Vector execution is excellent for optimizing the iteration overhead[32].

1.2 Problem Statement

From a host of available processors available today, databases can be maintained on heterogenous systems using various optimisation techniques. But their performance still leaves room for improvement. Tuning hardware for better speeds, and developing cheap solutions that can be effectively used for database management will be key as the volume of data under processing keeps rising. And finding optimal solutions for network-distributed systems will also be a step forward in the databases that have a constant stream of data.[11]

1.3 Our Vision

The goal is to amalgamate the performance enhancement of hardware tuning, and good software engineering practices to be applicable on database management systems. It would be ideal to find a universal solution, that can be implemented in contemporary systems as well as current DBMS so that scaling from one system or processor to another is hassle-free. In that regard, we will highlight the premise, advantages and shortcomings of various approaches in this paper. We try to propose a harmonious use of the latest programming advancements and see how database operations perform.



Processing Device	Parallelization Properties		Memory Scaling	
	Pipeline Parallelism	Data Parallelism	Memory Capacity	Memory Bandwidth
Central Processing Unit	+ / + +	+	+ +	+
Accelerated Processing Unit	+ / + +	+	+ +	+
Many Integrated Core	+ / + +	+ / + +	+	++
Graphics Processing Unit	—	+ / + +	+	++
Field-Programmable Gate Array	++	—/ +	—/ +	++

Legend: + + = excellent, + = good, — = poor

Figure 2: Comparison of Processing Capabilities of Assorted Processor Devices [5]

In the subsequent sections, we discuss current processing devices available, comparison of their performance, other upcoming approaches, and then advancements in network distributed systems.

2 POTENTIAL OF VARIOUS PROCESSING DEVICES

Hardware is consistently evolving and can be designed according to the needs of the user, or component. Since there are a number of processing devices, and each can be adapted and tuned to for diverse database operations, we would be comparing FPGAs, APUs, CPUs, and more to understand their design and implementation.

- **Many Integrated Core.** Multiple cores together can handle the bulk of processor-intensive operations if coded properly[26]. It can enhance the performance quite well in pipeline conditions for example. Intel developed their MIC architecture, in which each core can successfully process 4 hardware threads. The disadvantage of this option is that it can process data present only on its device memory. Cache shared between the multiple cores can also pose an overhead. In the Xeon Phi from Intel, there are also vector processing units that support a high volume of data to be parallel-processed.
- **Graphics Processing Unit.** Graphics Processing Units were designed specifically to handle the large amount of processor-intensive rendering etc, required in media and gaming industry. But the huge boom in the industry has resulted in the evolution of excellent GPUs[6]. They are also tuned to perform in intensive conditions. So evidently it made sense to exploit the processing capabilities of GPUs in database management systems. They can do very effective SIMD processing, but the data input/output express lines can pose a disadvantage. Parellel data effectively has to be serialised for good performance.
- **Multi-Core Central Processing Unit.** Since the CPU is a powerful processing unit, the idea of combining multiple cores into one Central processing unit is a popular solution[9]. In case the code is parallelisable, a high level of parallelism can be achieved since each core can process several threads at the same time.[22] Single instruction, multiple data SIMD instructions can be used to execute the branching conditions in a query too, to improve the performance. [9]
- **Accelerated Processing Unit.** The so-called accelerated processing unit (APU; also coupled CPU-GPU architecture) consists of a CPU with an integrated GPU. Consequently, an APU unites properties of both devices. Furthermore, the CPU and GPU in an APU use the same main memory as well as consolidated caches, which is a high benefit in contrast to autonomous CPU and GPU, because of the reduced cost of data transfer[18]. However, since chip space is limited, the GPU part in the APU has less cores and runs on a lower frequency than their autonomous counterparts.
- **Field-Programmable Gate Array.** FPGA is necessarily an interconnection between look-up tables, which increases the speed of the database operations performed on redundant, and large volumes of data. Through a boolean function, a field-programmable gate array can be aligned to any application according to the need. The logic units in FPGAs can be pipelined for better performance in a clock cycle, which is why they perform well in terms of pipeline parallelisation (data

too).[29] The only shortcoming be because of limited chip space.

3 COMPARING PROCESSOR PERFORMANCES

Since the underlying design is specific to most devices, they can be compared on the factors like the memory management capabilities, and the degree of parallelization that can be achieved.

3.1 Memory Scaling

Memory is an important factor for data intensive applications.

- *Memory Capacity*
  Specialised devices have comparatively less memory than devices that use the main memory. Moreover, data should be present at the accessible memory for efficient processing by the dedicated device. If the memory is less, processing can be slowed down especially in data intensive applications.
  Since CPUs and APUs can use the main memory, their memory capacity is good. Also, the installable RAM can be increased through the NUMA capabilities (Non-uniform memory access) of CPUs and APUs, which gives them excellent memory capacity. However, GPUs and MICs can use their memory, which is good, but less as compared to devices that have access to hundreds of gigabytes of main memory. FPGAs behave differently because they process data streams. The memory capacity ranges from poor to good depending on the type of FPGA.
- *Memory Bandwidth*
  Memory Bandwidth describes the data access speeds on the device memory. This is an important factor because bottlenecks due to memory[30] have become an important factor in-processing. CPUs and APUs have good memory bandwidth. But using NUMA to extend bandwidth has posed some problems[27]. And the bandwidth of GPUs, MICs and FPGAs is excellent as the memory is hardwired to their processing units.

3.2 Parallelisation Properties

- *Pipeline Parallelism*
  To execute a certain command, several instructions have to be executed which can be done in an ordered manner, akin to a pipeline. For efficient execution, the instructions can be pre-fetched and loaded. This is especially important in case of branching so that the correct sequential instruction is implemented. This process tries to avoid latency due to cache miss. This can occur when the next instruction is not loaded into memory when the previous instruction is successfully implemented. The CPU tries to predict the branch based on previous executions. These predictions tend to fail if the selection factor is 0.5 ie. the branch executes just as many times as it does not execute. Eventually, this can cause delays in the pipeline execution process[31].
  Based on the complexity of a given problem and the overall code, different processors perform variably. If branch predictions are accurate, CPUs can give excellent to good pipelining performance. Similarly, Accelerated Processing Units(APUs) and Multi-Integrated Core (MICs) offer good to excellent pipe-lining results because they are CPUs integrated with GPU, and multiple CPUs respectively.
  FPGAs perform excellent because they are pipeline programmed, which reduces the chances of a pipeline flush due to mis-prediction of a code branch. Also, data items can be moved in each clock cycle by having pipelined registers between FPGA logic units[29]. On the other hand, GPUs do have good pipeline parallelism because there is almost negligible control logic offered. But when a code has multiple branches, they can be serialised and then thread-executed via GPUs.

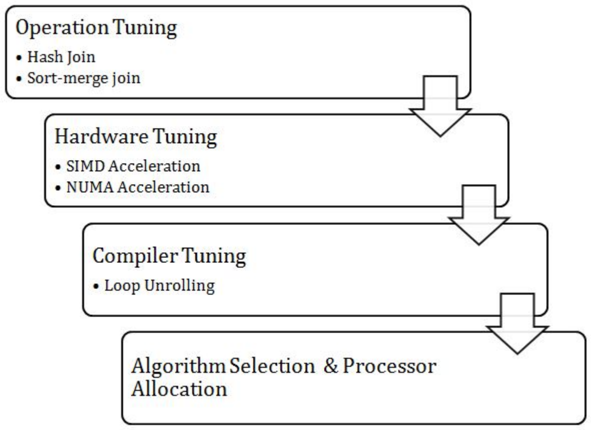
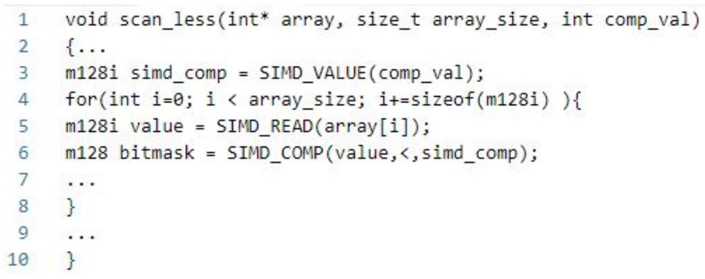


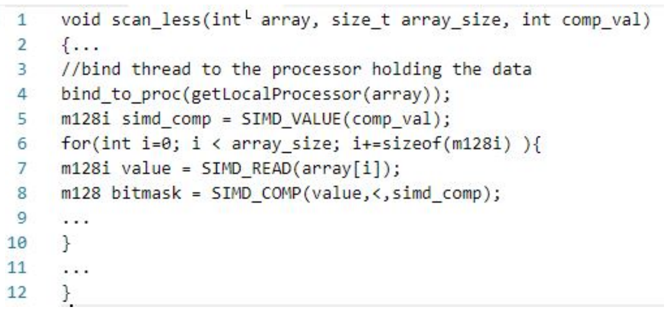
Figure 3: Hardware Tuning Process [5]

- *Data Parallelism*
  In this parallelism technique, the same instruction can be applied to different data blocks simultaneously. However, it has one caveat that the data should be independent and the result of one instruction must not be affected by another data item during the same clock cycle. The advantage of data parallelism is that it decreases data cache misses.



```
void scan_less(int* array, size_t array_size, int comp_val)
{...
m128i simd_comp = SIMD_VALUE(comp_val);
for(int i=0; i < array_size; i+=sizeof(m128i) ){
m128i value = SIMD_READ(array[i]);
m128 bitmask = SIMD_COMP(value,<,simd_comp);
...
}
...
}
```

Figure 4: SIMD Accelerated Scan [5]



```
void scan_less(int* array, size_t array_size, int comp_val)
{...
//bind thread to the processor holding the data
bind_to_proc(getLocalProcessor(array));
m128i simd_comp = SIMD_VALUE(comp_val);
for(int i=0; i < array_size; i+=sizeof(m128i) ){
m128i value = SIMD_READ(array[i]);
m128 bitmask = SIMD_COMP(value,<,simd_comp);
...
}
...
}
```

Figure 5: NUMA Aware SIMD Accelerated Scan [5]

For CPUs with AVX [28](Intel's Advanced Vector Extensions) and SSE-registers (Streaming SIMD Extensions) the data parallelism capability is good. But it still does not compare up to MICs and GPUs. Similarly for APUs that support data parallelism but not as well as dedicated GPUs. This is because there is limited space on APU chip which limits the number of cores that can be present on it. Conversely, there is good to excellent data parallelism possible on MICs and GPUs due to multiple cores that can be used for many partitions. On FPGAs the range of data parallelism can range from poor to good, depending on its configuration. The user can configure it, but there is always a trade-off between the data parallelism and pipeline parallelism. Essentially, pipeline parallelism is more effective than data, on FPGAs[29].

Conclusion Different processing devices differ immensely on their parallelization and memory capabilities. This comparison is summarized in Figure 1. Thus, these factors are very important to note for an efficient use of co-processors for database operations.[31]

4 CUSTOMISED DATABASE PROCESSING SOLUTIONS

It is now evident that to fully utilise different processing devices, we have to alter implementation of database operations according to the specific underlying hardware as shown in Fig 2. It is achieved by implementation of techniques, producing maintainable database operations, and solving the problems observed along the way.

4.1 Implementation

We highlight the amalgamation of different methods for achieving the highest level of possible performance.

The first step is to implement algorithms for different database operations, that have been produced by multiple authors[19][7][6]. Next, for each processing device a new *Flavor* has to be created which is a modified algorithm, best suited for that device. Third is Compiler Tuning because that will also impact the overall performance of the algorithm[1]. The fourth step is to make the database aware of these flavors that are available for various co-processors. Therefore, processor allocation and algorithm selection also play an important part in the overall performance[24].

4.2 Short-Comings

A simple way to tune hardware is to develop an algorithm for each processing device that is used in a processing device. But the cost involved in creating algorithms specific to each device by a different vendor would be very high in terms of implementation, and maintenance. Following are all things to be considered in the tuning process. as shown in Figure 3.

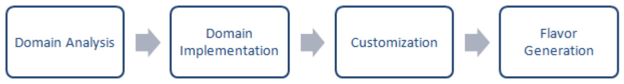
So, the focus has to be on two things: *Maximum Performance*: Algorithms should give the best performance while using the device capabilities.

Less Maintenance Effort: Less implementation and maintenance costs by pairing the same code snippets written for similar functionality and using that developing new flavors. Currently, we rely on OpenCL to create optimized code for various processing devices under research. But the hand-tuned algorithms perform better than ones produced using OpenCL[20]. Consequently, OmniDB[25] is a good alternative because it uses adapters to align the code with the hardware. But even with this, a new adapter has to be created with each device, still incurring an extensive manual implementation effort.

4.3 Proposed Solutions

Here we discuss that our vision can be implemented through Feature-Oriented Software Development, by using common code for database operations. This will cut down the implementation and maintenance efforts. This technique is already used for Software product lines (SPLs)[23]. Flavor is a name for a slighly modified algorithm. [2] *Domain Analysis*: Analyse the domain to find similarity with other domains, followed by developing a feature model with different implementation parts[23]. For example, in Figure 4 there is an example of SIMD accelerated scan[5] and in figure 5, there is an example of a SIMD accelerated scan with NUMA awareness.

Domain Implementation: Different code artefacts can be used from differing implementations to create code for this device. Some code transformation also occurs at this step.



Customization: Customising the code to the underlying hardware. *Flavor Generation*: Here, flavors generated are combined with selected features, customization and domain implementation step for final output.

5 DISTRIBUTED DATABASE SYSTEMS

A database where all storage devices are not attached to open processor is known as a Distributed database. They can be on multiple computers at one location, or connected online in a network of computers. There is a high degree of reliability and availability. When a large database is divided, each part becomes a small database in itself while retaining its characteristics, and runs on its hardware, software and transaction load. Therefore, it results in improved performance for queries as compared to a large database. Also, transactions that access more than one site can be done in parallel which is another way of decreasing the output time of the transaction. The individual results are combined later to display the final output.

5.1 Concurrency Control in Distributed Databases

Distributed databases are more complex because there is an added complexity of concurrency control because multiple copies are maintained at the different data sites. And there are different checks in place to ensure that the database is consistent while in use by many users and transactions. Early databases employed Timestamps as a measure of concurrency control, but there is a huge overhead of maintaining them.

These days most database management systems use the two-phase locking and two-phase commit techniques to reduce the number of rollbacks on committed transactions. In the first phase, the transaction sends a ’prepare to commit’ message to each locked copy. Each copy sends out a ’ready to commit’ message and once these messages are received from all affected copies, the transaction sends a ’commit’ message. If a ready message is not received from all copies, the transaction is rolled back at all sites.

Both the methods of concurrency control measures for distributed databases result in a large amount of network traffic among different database locations. And, there can be an issue wherever the locks were not removed by the database location because the commit message was not received. So, most distributed database systems allow the DBA(DB administrator) to set an idle time limit on the transaction. In case of a time out, the transaction is rolled back and does not restart.

5.2 Data Management Techniques

The new generation of distributed in memory DBMSs with RDMA (remote direct memory access) would need a new design because they are built under the assumption that a bottleneck exists at the network. But this assumption is no longer true [8]. For example, using the InfiniBand FDR 4×, the bandwidth for transferring the data is comparable to the bandwidth of one memory channel. Infiband is a networking standard used for high-performance processes and promises a high throughput and low latency. The RDMA transfer latencies have also been improving. Introducing a new architecture for this problem for OLTP and OLAP has shown a great improvement in performance as compared to traditional designs. It is not the case that network latency will exceed the memory latency, but that the efficient tuse of local memory and proper cache management will be even more crucial for small requests such as a hash table lookup because performance will be network transfer independent. Also, Infiband is getting cheaper for deployment. And the myth that distributed transactions cannot scale well, does not stand true[11].

5.3 Advancements in Distributed Systems

Earlier, it was said that distributed transactions cannot be scaled. But Zamanian et al, have developed a database system NAM-DB [8], which shows that transations with simple Snapshot Isolation are scalable using RDMA (Remote Direct Memory Access) network technology.

Network-Attached Memory

Compute and storage in the NAM architecture are decoupled logically. The storage servers have a shared memory pool, accessible from the compute nodes. The storage nodes oblivious to DBMS operations like joins, and consistency, etc are implemented by the compute nodes. This removes dependencies and the storage nodes can focus on garbage collection, metadata management(mapping the remote memory address to the required data page), data reorganisation, etc. The system has control over what data is copied and synchronised.

Other advantages of the NAM architecture are that the storage nodes can be scaled independently. It can handle well any data imbalance because each node can access data at different locations without the need for data redistribution. This is not a unique phenomenon because some systems us key-value interfaces[13][12], but have been built from scratch for increasing the distributed network performance. It is important to see that this architecture builds on the features of the underlying hardware and byte-level access to memory. The addressability also solves the issue of maintaining concurrency.

5.4 Mars: A MapReduce Framework[17]

The MapReduce framework works on two primitives. The map function is applied on every key-value pair(k1,v1) to output a list of intermediate key/value pairs (k2,v2). The reduce function is applied to all the intermediate values and gives a list of output vales. Now the programming logic can be added within these two primitives. System ensures the parallel execution of the two tasks, and it can be handled using APIs as per the requirement. Ever since the **MapReduce framework** was introduced by Google[10], it has found application in various domains like data mining, bioinformatics, etc. Hadoop is also an open-source implementation of the same framework. Yang et al.[16] had introduced merge operation of relational databases in the framework. **Phoenix**[4] was another implementation on a multi-core CPU. The Mars framework includes splitting the input data, map process through different threads, sorting, splitting and reducing in n threads and merging for final output. Finally, the result is copied from the GPU device memory to main memory. Different attributes include directory index, key and value input, intermediate key, and output keys. One GPU thread handles one chunk of data, the intermediate key value pairs are sorted so that pairs with same key are stored consecutively. Techniques like **thread parallelism**, **coalesced accesses, built-in vectors, file manipulation, handling variable input sizes** and **sort** are used correspondingly. .

They implemented six applications based on different core computations for web analysis for benchmarks of the framework including String Match and Inverted Index in web document searching, Similarity Score (**SS**) and Matrix Multiplication(**MM**) in web document processing, and Page View Rank (**PVR**) and Page View Count (**PVC**) for web log analysis. As compared to Phoenix, Mars is 1.5-16X times faster on large databases. The performance depends on the size of accessed data in each task. Foe example, the speed is high for PVC and MM because every map in these functions fetches long integer vectors, and the built in vector becomes beneficial.

6 GPUTERASORT

Govindraju et al. developed a new algorithm called GPUTeraSort for sorting more than a billion records wide databases using Graphics processing units[14]. Their algorithm utilises the both data and task parallelism on a GPU for compute intensive tasks while the CPU does the I/O and resource management. They have used the high bandwidth GPU interface and CPU main memory interface whihc is low bandwidth comparatively to attain more memory bandwidth than pure CPU-based processes. It is a pipeline that consists of two phases:

- Disk read, key building, sorting through the GPU, generating runs and writing disk.
- Reading, merging and writing

It will also consequently pipeline the disk transfer processes to achieve peak performance on a standard Sort benchmark. While testing GPUTeraSort on an NVIDIA 7800 GT 3GHZ Pentium PC, the performance was significantly better as compared to normal processing. It reported the best PennySort price-performance which prove that using GPU as co-processor can greatly improve performance on large databases and respectively large database tasks. They have achieved speeds 10x as compared to only CPU executions.

They have mapped a **bitonic sorting network** to the **rasterisation operations** in a GPU, and used the GPU's programmable hardware and the corresponding high-bandwidth memory interface. Their contributions consist of creating an external sorting architecture that divides the work amongst the CPU and GPU; creating an in-memory sorting algorithm for GPUs which is up to ten times faster than CPUs; nearing I/O peak performance and almost peak memory bandwidth on an affordable PC, and developing an approach that is scalable to very large wide-key databases for sorting immense data partitions.

Some **limitations** are that it is based on radix sort and so it works best only for keys with fixed bytes. Also, it does not benefit on almost sorted input data files. It also requires programmable GPUs with fragment processors, which some previous versions of GPUs may not have.

These large sort tasks have implementation in web indexing engines and other exemplary systems like the geographic information systems, data mining and super-computing etc. Huge sort tasks arise in many different applications including web indexing engines, geographic information systems, data mining, and supercomputing. Sorting is also a proxy for any sequential I/O intensive database workload. This article considers the problem of sorting very large datasets consisting of billions of records with wide keys

7 CONCLUSIONS AND FUTURE WORK

In this work, we argued for adjusting database systems processing according to the design and performance capabilities of the hardware. We saw the advantages, and disadvantages of numerous hardware devices, and how they would perform database operations. Since these devices were created for some other purpose entirely, it is clearly seen that the performance ranges from excellent to bad depending on the conditions under which testing is done. These metrics can only guide in the right direction from where database operations can be tuned to perform better.

The MapReduce framework can be extended to other applications and domains like data mning. The Mars framework specifically can be integrated to other implementations like Hadoop which can exploit the parallelism among different machines while using the machine parallelism as well. In case of network distributed databases, the Infiband research has opened up a research area of the trade off between remote processing and local processing, or how to create abstractions that cover the complexity of RDMA verbs. Also, the fact that messages in connected queues are ordered can be taken as advantage for further work[8]

Moreover, we saw that a combination of hardware, software tuning and feature-oriented software development can be key to designing good database systems on varied processor devices.

This is burgeoning research, and the efforts of different individuals, and latest technology in synchronicity will give us advanced, efficient methods for database management systems. And since network-distributed systems are also growing at a fast-pace, great optimization is on the horizon for those systems as well.

REFERENCES

[1] M. D. Hill A. Ailamaki, D. J. Dewitt and D. A. Wood. *DBMSs on a modern processor: Where does time go?* 1999.

[2] P. Boncz B. Raducanu and pages=In SIGMOD, pages 511–524 M. Zukowski, year=2013. *Micro adaptivity in vectorwise.*

[3] Galakatos A. Kraska T Zamanian E. Binnig C, Crotty A., 2016.

[4] C. Kozyrakis. Bradski. Evaluating mapreduce for multi-core and multiprocessor systems. symposium on high performance computer architecture (hpca). *SIGMOD*, 2007.

[5] Breß S. Heimal M. Saake G Broneske, D. Toward hardware-sensitive database operations, 2014.

[6] J. Teubner C. Balkesen, G. Alonso and M. T. Ozsu. *Relational joins on graphics processors.* 2008.

[7] J. Teubner C. Balkesen, G. Alonso and M. T. Ozsu. *Multi-core, main-memory joins: Sort vs. hash revisited.* 2013.

[8] A. Galakatos T. Kraska E. Zamanian C. Binnig, A. Crotty. The end of slow networks: It's time for a redesign, 2016.

[9] J. Chhugani T. Kaldewey A. D. Nguyen A. D. Blas V. W. Lee N. Satish C. Kim, E. Sedlar and P. Dubey. S. Sort vs. hash revisited: Fast join implementation on modern multi-core cpus., 2009.

[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

[11] T. Harris E. Zamanian, C. Binnig and T. Kraska. The end of a myth: Distributed transactions can scale, 2017.

[12] J. J. Levandoski et al. *High Performance Transactions in Deuteronomy.* CIDR, 2015.

[13] M. Brantner et al. *Building a database on S3.* 2008.

[14] Naga Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort. page 325, 2006.

[15] G. Graefe. *Encapsulation of Parallelism in the Volcano Query Processing System.* 1990.

[16] R. Hsiao H. Yang, A. Dasdan and D. S. Parker. Map-reduce-merge: Simplified relational data processing on large clusters. *SIGMOD*, 2007.

[17] Bingsheng He, Qiong Luo, Naga K Govindaraju, and Tuyong Wang. Mars : A MapReduce Framework on Graphics Processors. *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 260–269, 2008.

[18] M. Lu J. He and B. He. *Revisiting co-processing for hash joins on the coupled CPU-GPU architecture.* 2013.

[19] A. Kemper M.-C. Albutiu and T. Neumann. *Massively parallel sort-merge joins in main memory multi-core database systems.* PVLDB.

[20] H. Pirk S. Manegold M. Heimel, M. Saecker and V. Markl. Hardware-oblivious parallelism for in-memory column-stores., 2013.

[21] Kersten M. L. Manegold, S. and P. Boncz. *Database Architecture Evolution: Mammals flourished long before Dinosaurs became extinct.* 2009.

[22] M. Zukowski P. A. Boncz and N. Nes. *Hyper-pipelining query execution. In CIDR, MonetDB/X100:.* 2005.

[23] C. Kastner S. Apel, D. Batory and G. Saake. *Feature-oriented software product lines - Concepts and implementation.* Springer, 2013.

[24] H. Rauhe K.-U. Sattler E. Schallehn S. Breß, F. Beier and G. Saake. *Efficient co-processor utilization in database query processing. Inf. Sys.* 2013.

[25] B. He S. Zhang, J. He and M. Lu. *OmniDB: Towards portable and efficient query processing on parallel CPU/GPU architectures.* 2013.

[26] M. Saecker and V. Markl. *Big data analytics on modern hardware architectures: A technology survey.* 2013.

[27] B. Schlegel T. Kiefer and W. Lehner. *Experimental evaluation of NUMA effects on database management systems.* 2013.

[28] I. Muller T. Willhalm, I. Oukid and F. Faerber. *Vectorizing database column scans with complex predicates.* 2013.

[29] J. Teubner and L. Woods. *Data Processing on FPGAs. Number 35 in Synthesis Lectures on Data Management.* Morgan and Claypool Publishers.

[30] S. Manegold TP. A. Boncz and M. L. Kersten. *Optimizing database architecture for the new bottleneck: Memory access.* VLDB, 1999.

[31] J. Zhou and K. A. Ross. *Implementing database operations using SIMD instructions.* San Val, 2002.

[32] M. Zukowski. *Balancing Vectorized Query Execution with Bandwidth-Optimized Storage.* PhD thesis, CWI Amsterdam, 2009.