A Hardware-Oblivious Optimizer for Data Stream Processing

Constantin Pohl
supervised by Prof. Dr. Kai-Uwe Sattler
Technische Universität Ilmenau
Ilmenau, Germany
constantin.pohl@tu-ilmenau.de

ABSTRACT

High throughput and low latency are key requirements for data stream processing. This is achieved typically through different optimizations on software and hardware level, like multithreading and distributed computing. While any concept can be applied to particular systems, their impact on performance and their configuration can differ greatly depending on underlying hardware.

Our goal is an optimizer for a stream processing engine (SPE) that can improve performance based on given hardware and query operators, supporting UDFs. In this paper, we consider different forms of parallelism and show measurements exemplarily with our SPE PipeFabric. We use a multicore and a manycore processor with Intel's AVX/ AVX-512 instruction set, leading to performance improvements through vectorization when some adaptations like micro-batching are taken into account. In addition, the increased number of cores on a manycore CPU allows an intense exploitation of multithreading effects.

Keywords

Data Stream Processing, AVX, Xeon Phi, SIMD, Vectorization, Parallelism

1. INTRODUCTION

Technological advancement leads to more and more opportunities to increase application performance. For stream processing, data arrives continuously at different rates and from different sources. A stream processing engine (SPE) has to execute queries fast enough that no data is lost and results are gathered before the information is already outdated. A solution to achieve this is a combination of software optimizations paired with modern hardware for maximizing parallelism. It is difficult to find an optimal parametrization though, e.g. for the number of threads or load balancing between them. It gets even worse when different hardware properties come into play, like memory hierarchies or CPU core count.

For this paper, we consider different aspects and paradigms of parallelization that are applicable on data stream processing. In addition, first measurements on SIMD and multithreading realized on an Intel Core i7 and Intel Xeon Phi Knights Landing (KNL) with our SPE PipeFabric are shown. Our final goal is a full optimizer on a SPE capable of dealing with unknown UDFs in queries as well as with arbitrary hardware the system uses. Two consequential tasks arise from this.

- Exploitation of opportunities given by modern hardware, like wider CPU registers on Intel's AVX-512 instruction set, manycore processors with 60+ cores on a chip or increased memory size.
- Analysis of performance impacts by possible UDFs and operations, like computational effort or possible parallelization degree in case of data dependencies.

The rest of this paper is organized as follows: Next Section 2 is a short recapitulation about stream processing, possible parallelism and opportunities given by hardware. Section 3 summarizes related work done on SIMD parallelism, hardware-oblivious operations and stream partitioning in context of data stream processing. Our results are presented in Section 4, followed by Section 5 with future work. Section 6 with conclusions tops off this work.

2. PREREQUISITES

This section shortly summarizes requirements on stream processing and parallelization opportunities in addition to information on used hardware, like supported instruction sets.

2.1 Data Stream Processing

As already mentioned, high throughput and low latency are main requirements for stream processing. A data stream delivers continuously one tuple of data after another, possibly never ending. Queries on data streams have to consider that tuples can arrive at alternating rates and they get outdated after a while, because storing all of them is impossible. Operating with windows of certain sizes are a common solution for this property.

As a consequence, a query has to be processed fast enough to produce no outdated results as well as keeping up with eventually fast tuple arrival rates. Handling multiple tuples

Proceedings of the VLDB 2017 PhD Workshop, August 28, 2017. Munich, Germany.
Copyright (c) 2017 for this paper by its authors. Copying permitted for private and academic purposes..

at once through partitioning of data or operators exploiting parallelism possibilities is therefore a must.

2.2 Parallelism Paradigms

There are mainly three forms of parallelism on stream processing that can be exploited.

Partitioning. Partitioning can be used to increase speedup through splitting data on different instances of the same operator. Therefore every instance executed in parallel has to add its function on a fraction of data, increasing throughput of a query. However, additional costs for assigning data to instances and merging results afterwards arise, influencing the optimal partitioning degree.

Task-Parallelization. Operators of a query can execute in parallel if data dependencies allow such parallelism. A pipelining schema provides possibilities to achieve this, realized by scheduling mechanisms.

Vectorization. A single instruction can be applied on multiple data elements, called SIMD. For stream processing, some preparatory work is necessary to use this form of parallelism, like storing a number of tuples before processing them at once with SIMD support. On the one hand, it increases the throughput of a query while on the other hand batching up tuples worsens latency.

We focus on partitioning and vectorization, because task-parallelism is mainly a scheduling problem that is not of further interest at this point.

2.2.1 Partitioning

A speedup through partitioning is achieved mainly with multithreading. Each partition that contains operators processing incoming tuples uses a thread, which leads to challenges on synchronization or load balancing, especially on manycores, as shown by Yu et al. [6] for concurrency control mechanisms. Partitioning is a key for high performance when using a manycore processor, which provides support for 200+ threads at the cost of low clock speed. To investigate the right partitioning degree between reduced load on each partition and increased overhead from threads as well as an appropriate function for forwarding tuples to partitions additional observations have to be made. Statistics are a common solution, but can be far away from optimal performance in worst cases.

2.2.2 Vectorization

To use vectorization, certain requirements must be fulfilled. Without batching up tuples it is impossible to apply a vector function on a certain attribute. This leads to the next challenge, the cache-friendly formation of a batch. Without careful reordering, any vectorization speedup is lost through expensive scattered memory accesses. A possible solution for this is provided by gather and scatter instructions that index through masking certain memory addresses for faster access. Additional requirements on vectorization arise through the used operator function and data dependencies between tuples. The function must be supported by used instruction set, while dependencies are solved through rearrangements or even fundamental changes on the function of the operator.

Figure 1 shows the processing model of a query with batching. Tuples arrive one at a time on the data stream, being gathered first on a batching operator with attribute grouping in memory realized by vectors until batch size is reached and then forwarded to the next operator.

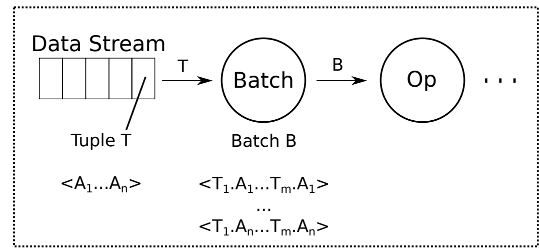

Figure 1: Processing Model

2.3 Hardware Opportunities

There are mainly two different ways to increase computational speed on hardware, distributed and parallel computing. Distributed computing uses usually many high-end processors connected to each other, sharing computational work between them. The disadvantage comes with communication costs. With requirements of low latency, we focus on parallel computing. Manycore processors like the Xeon Phi series from Intel use simpler cores, but many of them inside their CPU. This eliminates most of the communication costs, improving latency while providing wide parallelization opportunities compared to a single multicore processor.

The latest Xeon Phi KNL uses up to 72 cores with 4 threads each, available through hyperthreading. In addition, the AVX-512 instruction set can be used for 512bit wide operations (SIMD). This leads to great possibilities on partitioning and vectorization to reduce latency of a query. There are more interesting improvements on KNL like clustering modes, huge page support or high-bandwidth memory on chip (MCDRAM) which we will address in future work.

3. RELATED WORK

For parallelism through vectorization and partitioning on data streams, a lot of research has been done already, especially since manycore processors are getting more and more common. To achieve performance benefits, those manycore CPUs rely massively on SIMD parallelism and multithreading for speeding up execution time.

For data stream processing, the functionality of operations like joins or aggregations to give an example, are basically the same. However, for realization the stream processing properties have to be taken into account.

Polychroniou et al. [4] take a shot on different database operators, implementing and testing vectorized approaches for each one. Results show a performance gain up to a magnitude higher than attempts without SIMD optimization. In addition to this, their summary of related work gives a good review about research done with SIMD on general database operations.

For stream partitioning, the degree in terms of numbers of partitions as well as the strategy like the used split function for data tuples are the main focus of research. Gedik et al. [2] visited elastic scaling on stream processing where parallelization degree on partitioning is dynamically adjusted on

runtime, even for stateful operations. They reviewed typical problems when auto-parallelization is used like in most of other approaches.

For an optimizer that is hardware-oblivious, additional points must be considered. Hardware-oblivious means, that the optimizer is able to maximize performance on any hardware used, e.g a multicore or a manycore processor. Heimel et al. [3] implemented an extension for MonetDB called Ocelot, which is a proof of concept for hardware-oblivious database operators. They show that such operators can compete with hand-tuned operators that are fitted exactly for used processing units, like CPUs and GPUs. Teubner et al. [5] attended to the same topic before, looking deeper into hardware-conscious and hardware-oblivious hash joins.

4. EXPERIMENTS

With our experiments, we want to show the grade of impact on vectorization and multithreading for data stream processing. We therefore use two different processors, an Intel Core i7-2600 multicore CPU as well as an Intel Xeon Phi KNL 7210. As already mentioned before, KNL supports AVX-512 with 512bit register size and 256 threads, in contrast to i7s AVX with 256bit and 8 threads.

4.1 Preliminary Measurements

First measurements in Table 1 show needed runtime for corresponding CPUs when vectorization is enabled or disabled. Therefore an array with 64*1024*1024 elements is traversed, applying an addition operator (using 32bit precision) or square root operator (using 64bit precision) on each of the elements.

With vectorization, the speedup gain ideally corresponds directly with the number of elements processed at once, e.g. when using 32bit integers and the register size is 512bit, 16 elements are processed with one operator execution, leading to an expected 1/16th of runtime. However, this is not the case, because these elements needed to be accessed in memory (ideally in cache). With increased complexity (in terms of CPU cycles) this accessing costs are reduced compared to operators costs, as it can be seen in Table 1 with simple addition and complex square root. When computing the root, vectorization effectively doubles the execution speed on i7 processor and even more on KNL. On KNL, the registers can hold up to 8 64bit floating point numbers, resulting in around eight times faster execution on square root. On addition, however, even with prefetching mechanism it is not possible to pull data fast enough into the registers, because a simple addition just uses one CPU cycle. Therefore the full speedup cannot be achieved.

Results on square root on i7 processor can be explained through underlying hardware. Ideally, with AVX, 256bit registers and 64bit numbers, the speedup should result in around four times faster execution, however, it is only doubled. Further research points to how 256bit register are realized on i7-2600 (Sandy Bridge) - as processor of the first generation of AVX instruction set, it still uses two 128bit registers combined to achieve 256bit width. On performance there is only a small benefit of using 256bit loads and stores compared to 128bit, leading to only doubled speedup.

Processor	Vectorization	Addition	Square Root
i7-2600	disabled	42ms	187ms
	enabled	30ms	92ms
KNL 7210	disabled	98ms	998ms
	enabled	40ms	129ms

Table 1: Vectorization Runtime
Traversing array of 64*1024*1024 elements

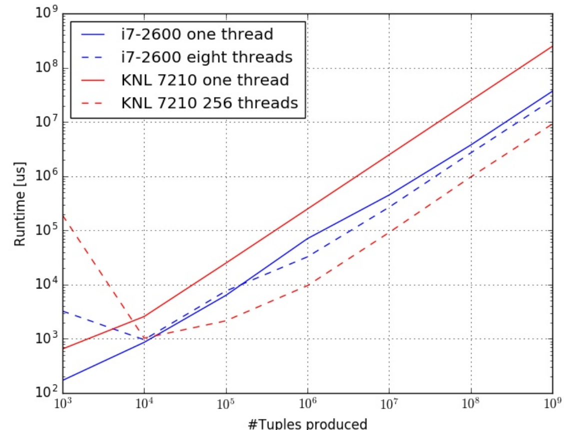

Figure 2: Query with Multithreading

4.2 SPE Tests

Our SPE PipeFabric is a framework for data stream processing, written in C++. The data streams as source of tuples can be constructed for example through network protocols. A query consists out of different stream processing operators that combined are forming a dataflow graph. It supports selections, projections, aggregates, groupings, joins and table operations, as well as complex event processing. The focus of the framework lies on low latency, realized through efficient C++ template programming.

For the tests, the data stream produces tuples through a generator. Increasing the number of attributes or using different data types just add a constant delay for each tuple, increasing runtime without changing the curves significantly. Therefore only a single integer as an attribute is counted up. In Figure 2, the needed time to produce a certain amount of tuples (up to 10^9) is measured (note the logarithmic scale of y-axis). With low number of produced tuples, the overhead through thread generation worsens execution time. This changes very quick, providing an intense speedup. When generation is singlethreaded, KNL performs worse than i7-2600 caused by slower clock speed. But when cores are fully utilized, running maximum number of threads through OpenMP, KNL can outperform the multicore CPU easily.

Figure 3 shows speedup achieved on i7-2600 and KNL on queries with vectorization. Therefore tuples are batched first with different batch sizes on each run, followed by an aggregation operator which applies a simple addition or a complex square root on single attributes. These aggregations are performed with and without vectorization, the difference on runtime results directly into speedup, e.g. when runtime is halved, the speedup is 100%.

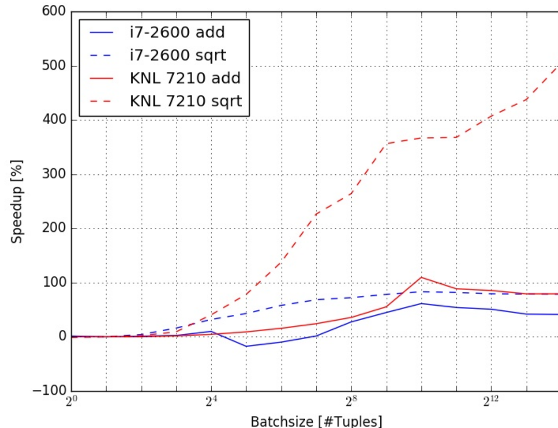

Figure 3: Query with SIMD

Taking a further look on Figure 3 reveals that speedup increases with batch size. This is relatively obvious, because with more tuples that can be processed at once by increased throughput, runtime of the query gets lower. However, this comes with the cost of latency, because results are delayed until a batch is full. An additional observation between addition and square root as aggregation operator can be made. With addition function, the performance gain is relatively low. This is because accessing the batch in cache takes longer than applying vectorized addition on it, even with prefetching mechanism. With square root, the operation takes significantly longer (in CPU cycles), so it is not limited that much by memory access on cache anymore.

5. RESEARCH PLAN

Vectorization and partitioning are the main two strategies which provide the most performance gain when set up accordingly to stream and query properties. Regarding vectorization, a batching mechanism is needed to exploit the full parallelism of wider CPU registers. With increased batch size, results of the query are delayed leading to increased latency but higher throughput. For partitioning, too many or too few partitions apart from optimum can even worsen the query execution time, same with uneven load balancing between partitions as shown by Fang et al. [1]. It is all a matter of right parametrization, depending on query, operators and underlying hardware.

This leads to our future work, where we will analyze impact on performance for certain strategies to finally come to an hardware-oblivious optimizer for queries on data stream processing, capable of dealing with UDFs as well as with different hardware sets. In difference to hardware-conscious optimizers which deliver only good precision for optimizations on certain hardware, our optimizer should generally being able to adapt its strategy on any given modern hardware.

However, it is a tradeoff between speedup and latency, greatly influenced by hardware used. We focus on parallelization on multicore and manycore CPUs, especially the latter one, because it is the most promising architecture for performance increasements with given requirements. With technical advancement additional chances are given, e.g. memory on package with high bandwidth, called MCDRAM on the latest manycore Xeon Phi KNL processor. When UDF support is realized, it is necessary to investigate key parameters for optimization, e.g. complexity of used function and data dependency.

6. CONCLUSION

For data stream processing, high throughput in terms of being able to process as many data as possible at the same time as well as low latency with fast responses on queries are main requirements. Exploiting parallelism is the answer, which is possible at different levels and degrees. In this paper, we show influence of parallelism on instruction level with vectorization as well as multithreading with our SPE PipeFabric and compare first results between multicore and manycore CPU.

SIMD effects improve performance when data is stored in a cache-friendly way within contiguous memory. For stream processing, each tuple cannot be processed one after another, therefore a batching mechanism is needed. This batching has to take care of storing data carefully for SIMD processing. Increased register width of Intel's Xeon Phi KNL with AVX-512 support leads to significant performance gains when not blocked by slow memory accesses. On the one hand, the computational workload must be high enough to surpass memory or cache accesses. This is not the case when additions on an aggregation operator are performed, as we showed in our experiments. On the other hand, with increased complexity SIMD operations are difficult to realize and must be supported by used instruction set.

Multithreading is another important factor when it comes to a manycore processor. Slow clock speed leads to poor singlethreaded performance compared to a multicore processor. This disadvantage is negated when enough cores can be utilized and parallelism is maximized. However, communication between threads, memory accesses and scheduling from threads to cores are no trivial tasks for optimizing performance, therefore more measurements are needed to prove results.

7. REFERENCES

[1] J. Fang, R. Zhang, T. Z. Fu, Z. Zhang, A. Zhou, and J. Zhu. Parallel Stream Processing Against Workload Skewness and Variance. *CoRR*, 2016.

[2] B. Gedik, S. Schneider, M. Hirzel, and K.-L. Wu. Elastic Scaling for Data Stream Processing. *IEEE'14*, pages 1447–1463, 2014.

[3] M. Heimel, M. Saecker, H. Pirk, S. Manegold, and V. Markl. Hardware-oblivious Parallelism for In-memory Column-stores. *VLDB*, pages 709–720, 2013.

[4] O. Polychroniou, A. Raghavan, and K. A. Ross. Rethinking SIMD Vectorization for In-Memory Databases. SIGMOD, pages 1493–1508, 2015.

[5] J. Teubner, G. Alonso, C. Balkesen, and M. T. Ozsu. Main-memory Hash Joins on Multi-core CPUs: Tuning to the Underlying Hardware. ICDE, pages 362–373, 2013.

[6] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores. *VLDB*, pages 209–220, 2014.