Handling Matrix Calculations with Microservices within Scenarios of Modern Mobility

Malte Zuch, Andreas Hausotter, Arne Koschel
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science,
Hannover, Germany
email: Malte.Zuch@hs-hannover.de

***Abstract*—In the context of modern mobility, topics such as smart-cities, Car2Car-Communication, extensive vehicle sensor-data, e-mobility and charging point management systems have to be considered. These topics of modern mobility often have in common that they are characterized by complex and extensive data situations. Vehicle position data, sensor data or vehicle communication data must be preprocessed, aggregated and analyzed. In many cases, the data is interdependent. For example, the vehicle position data of electric vehicles and surrounding charging points have a dependence on one another and characterize a competition situation between the vehicles. In the case of Car2Car-Communication, the positions of the vehicles must also be viewed in relation to each other. The data are dependent on each other and will influence the ability to establish a communication. This dependency can provoke very complex and large data situations, which can no longer be treated efficiently. With this work, a model is presented in order to be able to map such typical data situations with a strong dependency of the data among each other. Microservices can help reduce complexity.**

Keywords–e-mobility; microservices; matrix calulations; workload decomposition

I. INTRODUCTION

In the context of modern mobility, there are often m to n assignments. In the field of e-mobility, m-vehicles are often considered in relation to n-charging points. Or for the real-time parking optimization, m-vehicles have to be optimized in relation to n-parking lots. Particularly in Car2Car-Communication, large swarms of vehicles have to be considered, in order to be able to identify all possible interaction points.

The basic data model is expressed in m-x-n or m-x-m matrices. The data are thus represented as normal adjacency matrices. These matrices can provoke large and complex data situations. By matrix multiplication of such matrices, additional data can be calculated which can be used in the area of Car2Car-Communication. This is the motivation of this work and will be explained in more detail below.

A. Motivation

For the analysis and optimization of scenarios of Car2Car-Communication, it is necessary to capture the whole data situation at first. It must be determined which vehicles have a favorable distance to other vehicles, in order to establish a communication bridge to each other.

For example, in Fig. 1 the distance from vehicle A to C may be too large to communicate with each other (Situation 2). However, communication from A through B to C could be enabled via a vehicle B close enough to vehicle A and C (Situation 1). In order to identify such a possible communication jump (and also 'deeper' jumps over several vehicles in between), large matrix multiplications are required, which

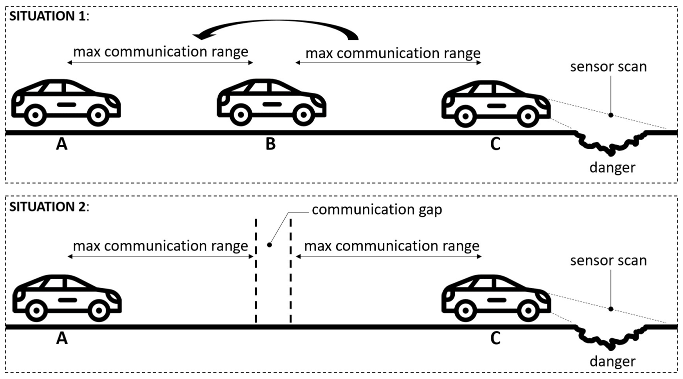


Figure 1. Situations of Car2Car-Communication

can demand a lot of memory and cpu time. The next Section explains what this work will contribute concerning this topic.

B. Contribution

The focus of this work is to deal with such large and deep matrix multiplications, which are often necessary within modern mobility scenarios and during the preparation of required data. These deep matrix multiplications can provoke a large demand for memory and cpu time and can exceed gives system capacities. Therefore, it is to be investigated how this large matrix multiplication problem can be treated more efficiently for applying of fine granular architecture concepts such as microservices. In this work, only the basic concept is presented, how calculation tasks of Car2Car scenarios can be decomposed. In the future, this decomposition could easily be implemented with microservices. In this work the mathematical model is explained and the microservices are only prototypically implemented in MATLAB [1] to illustrate the basic concept. For real applications, better runtime environments should be chosen. MATLAB was chosen only because it allows a very efficient matrix modeling. At first, the related work on this subject is shown in the Section II. In Section III the general data concept is presented and how this could be treated by the use of microservices. Further on, Section IV shows a number of exemplary measurements and how the computation resource demands can be reduced with this concept. The final Section V will summarize the results obtained and give an outlook on future work. But firstly, the next Section presents the related work on this topic.

II. PRIOR AND RELATED WORK

A general concept for a coordination system for optimizing the charging decision of electric vehicles were given in [2]. It was noted that concepts with parallel data processing would be necessary in the future in order to reactively cope with the coordination requirements of one million electric vehicles in Germany. In this respect, matrix models can be helpful in order to be able to partition the resulting data volumes. This was show in a prior work [3]. On the basis of these matrix models, optimization methods can help to assist electric vehicles in the search for electric charging stations. In an other previous work, first simulation results could show that vehicles can be supported theoretically and mutual blockages on load columns can be reduced. First synthetic simulation results show that up to plus 59 % more vehicles could be coordinated to electric charging stations [4].

A detailed overview concerning related matrix computation is given in [5]. A more detailed explanation of parallel matrix computation of thinly populated sparse-matrixes is given in this [6] related work. These sparse matrices could be used to reduce the volume of data. However, this is not part of this work. Here, the decomposition concept for the application of microservices will be presented. But these sparse matrix representations could help to understand modern matrix models and how they could be parallelized with regard to microservices. A general survey to microservices is presented in [7] and gives an overview of modern companies like Uber, Netflix, Amazon and Twitter using microservices.

The transition from monolithic application to modern microservices is shown in [8][9]. Both works give a good overview on the scaling of micro-services. This could be used in order to solve also large data problems, which among other things in the matrix models of this work are considered.

A recent survey of more than 100 IT companies has shown that only 20 % of companies are not considering microservices in their company decisions. For 80 % of the companies surveyed, microservices are already integrated or are currently in the integration process [10]. A similar survey confirms these findings, with 23.9 % of companies not yet in contact with microservices and the majority with 76.1 % are already in use of practicing or implementing microservices [11].

After this overview of the prior and related work, the general adjacency matrix model will be explained in the following Section.

III. The concept for handling huge matrices

This section deals with the core content. This is divided into two Sections. In Section III-A, fundamental matrix multiplications are addressed and how they are needed in modern mobility scenarios in order to be able to calculate dependencies and grouping dynamics of vehicle fleets. After the general calculation problem has been explained, Section III-B shows how this problem can be partitioned.

Large matrix multiplication often requires a lot of computing power. A linear increase in the number of columns and rows often results in a quadratic increase in the elements that must be calculated. Therefore, it makes sense to solve the problem from a certain problem size in sub-problems. These sub-problems can be solved, for example, with the concept of microservices. The general problem of those matrix multiplication in modern mobility scenarios is described in the following Section.

A. The definition of the general calculation problem

In many modern mobility scenarios, it is necessary to determine which vehicles have a small distance from each other and can form a local group. This is required, for example, in intelligent parking management system.

The larger the amount of vehicles in the same area, the harder the situation of competitive in finding free parking lots. Matrix multiplications of the adjacency matrix which contains the vehicle distances can be used to calculate where particularly strong competition situations occur. Car2Car-Communication especially is a even bigger problem. If it is to be calculated in real-time, which vehicles are currently close enough to form a cluster of Car2Car-Communications, this requires multiple matrix multiplications. So, in general, these multiple matrix multiplications are required in different modern mobility scenarios such as Car2Car-Communication or intelligent parking systems. Only for the example of Germany with about 45.8 million vehicles [12], this results in a large matrix multiplication with 48.5 million x 48.5 million entries.

Matrix multiplications can thus be used to determine, which vehicles can form a communication cluster together. This information can, for example, be used to initially exchange and aggregate sensor data among the vehicles in the cluster. Then, only a single vehicle provides all aggregated data via cellular connection to other clusters. This allows not all vehicles to communicate redundant information, but the information is only reported once per cluster. This can save bandwidth and relieve communication networks, especially if in the future millions of vehicles can send huge data in real-time. This scenario is shown in Fig. 2:

In order to identify dynamic clusters with matrix multiplications, a quadratic m-x-m matrix M is initially created in the dimension of the amount of m-vehicles. As soon as vehicles are below a maximum allowed communication distance r_{max} between each other, the matrix M receives a 1-entry. If vehicles are spaced apart from this distance, a 0-entry is made. This marks all vehicles that are too far apart to communicate directly with each other. Fig. 3 shows this:

This simple adjacency matrix M thus shows only the direct connections. As vehicle A is in communication range to B and B to C. D is not in range to any vehicle. But it is not directly apparent that an indirect connection of A over B to C is possible. This can be determined with matrix multiplications of the quadratic adjacency matrix M, containing the connections between the vehicles. The variable n describes the 'jump depth'. For example, for n = 2 it is possible to check whether vehicle A is indirectly connected to C via B.

$$Z_{tmp} = \sum_{i=1}^{n} M^i \tag{1}$$

As soon as an entry in Z_{tmp} is greater than 1, exactly 1 is entered at the same position in Z. Otherwise it is entered value 0:

$$Z_{i,j} = 1 \; if \; Z_{tmp_{i,j}} > 0 \tag{2}$$

$$Z_{i,j} = 0 \; if \; Z_{tmp_{i,j}} = 0 \tag{3}$$

$$for \; all \; i,j = [1,...,m] \tag{4}$$

$$m: \; amount \; of \; vehicles \tag{5}$$

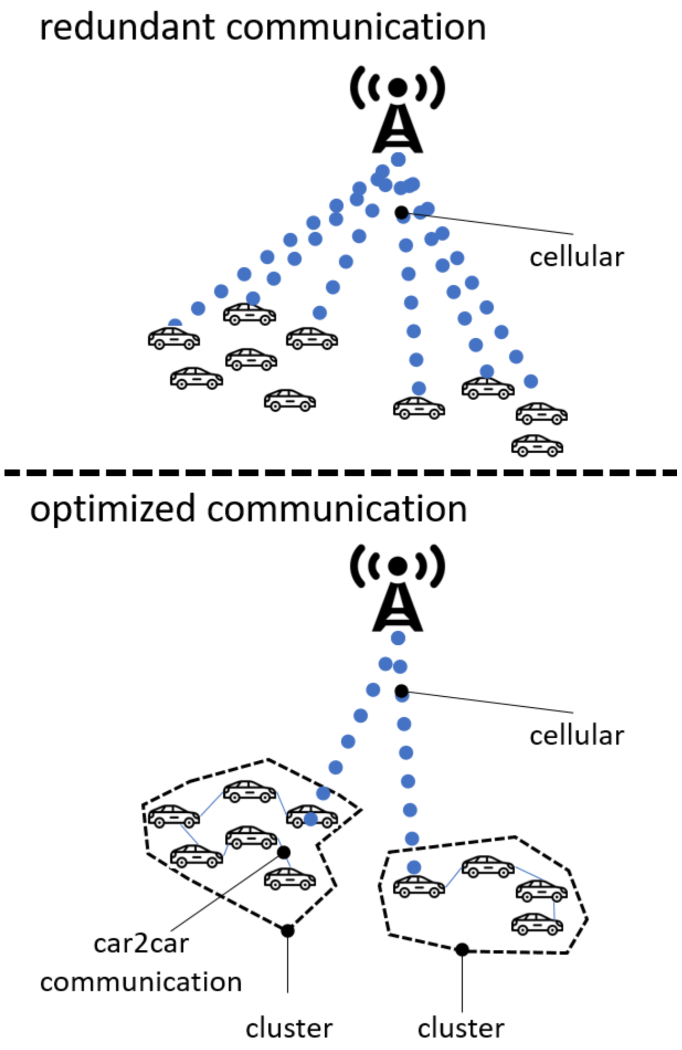


Figure 2. Scenario of Car2Car-Communication

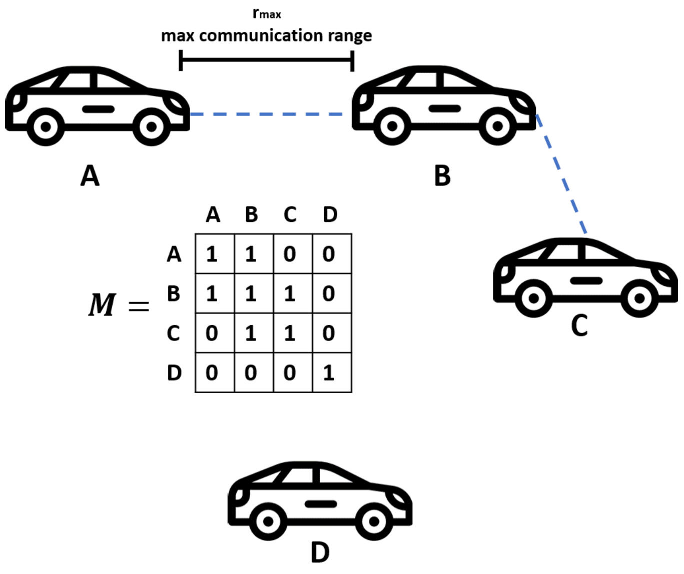
The result of those calculation from M over Z_{tmp} to the final cluster-matrix Z is shown in Fig. 4. In this example, only $n = 3$ 'jumps' were calculated. So, it was calculated how indirect jumps over other vehicles will lead to a cluster connection.

Fig. 4 shows, that there is no connection for D to any other vehicle. But it shows, that there is a indirect connection between A and C. This information was not directly available in the primary matrix M. This new information in Z can now be used to optimize scenarios such as in Fig. 2.

With very large matrices, a very bulky and big problem arises. For the calculation of scenarios that take into account the entire German vehicle market, this would quickly lead to a large demand for computing power. In this regard, the next Section explains how this problem can be decomposed in order to solve it more efficiently in parallel (for example, with the future use of microservices).

B. *The decomposition of the problem*

For a simple matrix multiplication of 4x4 adjacency matrix $M_{tmp} = M*M$, many intermediate steps must be performed.



$$M = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 1 & 1 & 0 & 0 \\ B & 1 & 1 & 1 & 0 \\ C & 0 & 1 & 1 & 0 \\ D & 0 & 0 & 0 & 1 \end{array}$$

Figure 3. The primary distance matrix

For example, to calculate the element $M_{tmp_{1,2}}$, the following sub-calculations are needed:

$$M_{tmp_{1,2}} = M_{1,2}*M_{2,1}+M_{2,2}*M_{2,2}+M_{3,2}*M_{2,3}+M_{4,2}*M_{2,4} \quad (6)$$

In the case of very large matrices, the entire calculation effort should then be decomposed. Fig. 5 shows this exemplary.

A slightly more visual representation is provided in Fig. 6. This shows how parts of the original matrix can be broken down into different tasks.

For example, the two tasks, TASK_A and TASK_B could very easily be represented as microservice, as shown in Fig. 5. Particularly in the case of very large matrices, thousands of micoservices can solve the cluster building problem in parallel. Each microservice only needs a fraction of the system resources that would be required for the entire problem. The mathematical modeling presented here thus makes it possible to easily map the entire (huge) computing problem for small microservices.

In the next Section, this concept will be examined. Some measurements are made, in which the cpu time demand is documented. It will show how the decomposition of the general calculation problem in smaller sub-problems can help to save computing resources.

IV. Measuring the Effects

At first, the computing resources (CPU Time) for solving the total and undivided matrix multiplication is measured. The matrix has the dimension 1024 x 1024 and the depth of computation was $n = 3$ (see (1) and Fig. 4). It is therefore calculated which vehicles (indirectly) can build a Car2Car-Communication with each other. Indirect means that vehicles build a communication bridge over other vehicles that are close enough together in the cluster. Such a communication bridge over other vehicles is indicated in Fig. 3. For purposes of better illustration, only one communications bridge over a single vehicle is illustrated in Fig. 3. However, the following calculation considers communication bridges for a deeper

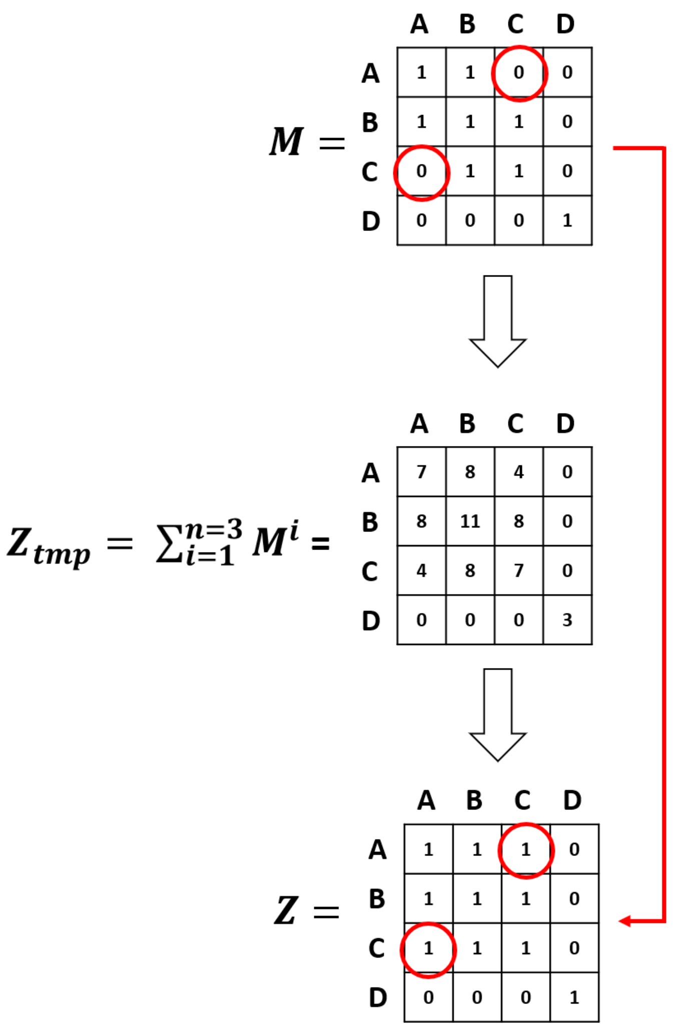


Figure 4. The result of the matrix multiplication

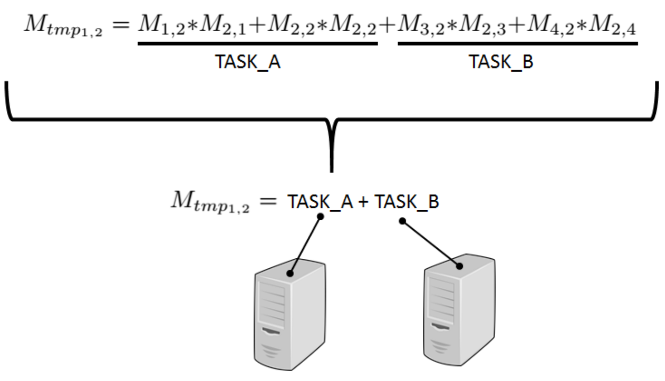

$M_{tmp_{1,2}} = M_{1,2}{*}M_{2,1}{+}M_{2,2}{*}M_{2,2}{+}M_{3,2}{*}M_{2,3}{+}M_{4,2}{*}M_{2,4}$
TASK_A
TASK_B
$M_{tmp_{1,2}} =$ TASK_A + TASK_B

Figure 5. The decomposition of a matrix multiplication

calculation of a total of 3 vehicles. This is the calculation depth $n = 3$. The hardware corresponds to current standard hardware from the year 2017 with a 4x3.30 Ghz processor and 8 GB Ram.

Next, the problem is decomposed into several equal sub-problems as mentioned in Fig. 5. The parallel basic prototype

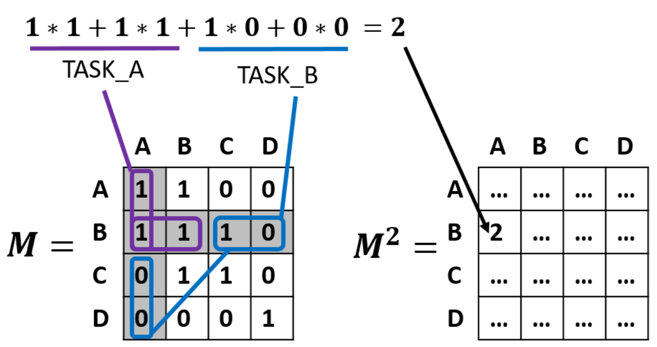


Figure 6. A example of matrix decomposition

microservices were realizes with MATLAB R2016b 64bit [1]. In future work, more common development environments and standards in designing microservices could be used [13] [14]. But for this first measurements of the general potential of partitioning, this more mathematical approach gives a quite good overview.

The dividing of a huge problem into smaller sub-problems reduces the general demand for computing resources per calculation instance, but also increases the communication overhead between all instances and will lead to a slightly longer time to calculate the problem. Depending on the cost of the microservices, the problem size and requirements with regard to real-time calculation, an different partitioning of the problem situation can be useful.

Table I gives a first impression, how different partitioning tasks affect the demand of computing resources (CPU Time) and the communication overhead (Communication Time).

TABLE I. THE EFFECT OF DECOMPOSITION

Partitions	*CPU Time*	*Communication Time*	*Total Time*
1	70.41 sec	0.0 sec	**70.41 sec**
2	32.09 sec	1.37 sec	**33.47 sec**
4	16.29 sec	2.48 sec	**18.77 sec**
8	8.38 sec	4.55 sec	**12.93 sec**
16	4.53 sec	8.84 sec	**13.38 sec**
32	2.60 sec	17.97 sec	**20.58 sec**
64	1.64 sec	36.32 sec	**37.97 sec**
128	1.13 sec	69.72 sec	**70.86 sec**
256	0.85 sec	138.97 sec	**139.83 sec**

The results from Table I were visualized in Fig. 7. It can be seen that a very high parallelization does not necessarily have to lead to a faster calculation.

It can be seen that with higher partitioning, the computation time per task decreases, but the communication effort to compose the final solution increases continually. With the used hardware (4x3.30 Ghz CPU and 8 GB Ram) the problem could be calculated fastest, if it is divided into eight sub-problems for the parallel calculation. This is marked in Fig. 7 with the dotted line. The optimal partitioning always depends on the hardware used. For the hardware used here, the optimum for partitioning is eight sub-problems. Cloud computing infrastructures with a large number of parallel processors would lead to even faster parallel calculations. But these first measurements can already show that a parallelization of matrix multiplications can lead to a efficient speed up.

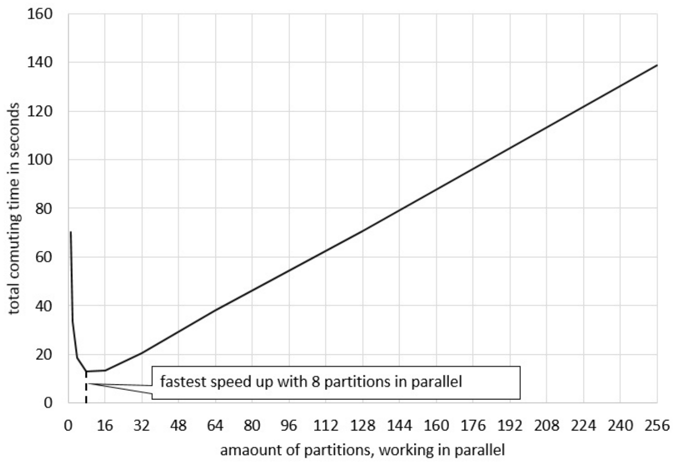


Figure 7. Visualization of the measurements

After presenting these first measurements, the next Section is followed by the discussion and outlook.

V. CONCLUSION

Without any partitioning, the total computing time was 70.41 sec. The best speed up was realized with a parallelization of 8, resulting in a total computing time of 12.93 sec. This is a speed up factor of 5.44x. A very fine partitioning with 256 partitions in parallel results in a total computing time of 138.83 sec. This is nearly two times slower than the initial computation time without any parallelization. So, the speed up factor was only 0.50x. This shows that the parallelization must be carefully chosen depending on the general size of the problem situation and the available hardware, in order to benefit from the optimal acceleration.

There are several parallelization possibilities. For example, the matrix multiplication could also be processed in parallel with typical MapReduce methods [15]. However, in the case of the MapReduce method, the coordination effort resulting from the map-layer and the reduction-layer, and the allocation to key-value pairs, can lead to somewhat more communication effort. But especially the matrix multiplications are characterized in particular by the fact that these can be split very easily. No prior sorting and assignment of the individual tasks to key-value pairs is necessary. The parallelization of matrix multiplication can be implemented very easily, without any key-value pairing and mapping-assignments (shown in Fig. 5). Therefore, the decomposition of huge matrix multiplication in very simple microservices seems to be appropriate.

Depending on the available computing power per microservice and the size of the overall problem, there is an optimum for the parallelization. With regard to the problems discussed here, it has been shown that the optimum of parallelization and additional communication effort is the fastest with a partitioning of 8 parallel tasks. Then the combination of additional communication effort and the acceleration of the parallelization is the fastest. With increasing parallelization, the pure calculation is still accelerated, but the entire processing time is reduced by the increasing communication effort, when the partial solutions are combined into the total solution.

In future work, the execution of the microservices with strongly optimized frameworks like Apache Hadoop / Spark [13] [14] could be investigated. The communication effort could be further reduced and even faster speed up factors could be reachable. So, summarizing, a parallelization of matrix multiplication (for example with the use of very simple microservices) can lead to a fast reduction in the computation time, but must be carefully chosen in order to achieve maximum efficiency.

REFERENCES

[1] "The MathWorks, Inc." 2017, URL: https://mathworks.com/products/matlab.html [accessed: 2017-12-06].

[2] M. Zuch, A. Hausotter, and A. Koschel, "Efficiency in the electromobile mass market (Original German Title: Effizienz im elektromobilen Massenmarkt)," INFORMATIK 2015 – 45. Jahrestagung der Gesellschaft für Informatik, Cottbus (DE), vol. 45. Jahrestagung, 2015, pp. 5–6.

[3] M. Zuch, A. Koschel, and A. Hausotter, "Partitioning model for mobile electric vehicle data," Digital Marketplaces Unleashed - Mobility Services, vol. IEEE 2016, 2017, pp. 1–3.

[4] C. Linnhoff-Popien, R. Schneider, and M. Zaddach, Eds., Digital Marketplaces Unleashed. Springer, Berlin, Oct. 2017, ISBN: 978-3-662-49274-1.

[5] H. G. Gene and C. F. v. L., Eds., Digital Marketplaces Unleashed. J. Hopkins Uni. Press, Dec. 2012, ISBN: 978-1421407944.

[6] J. P. Gray, Ed., Parallel Computing: Technology and Practice. IOS Press, Apr. 1995, ISBN: 978-9051991963.

[7] R. RV, Ed., Spring Microservices. Packt Publishing, Birmingham, Jun. 2016, ISBN: 978-1786466686.

[8] S. Sharman, R. Rv, and G. D., Eds., Microservices: Building Scalable Software. Packt Publishing, Birmingham, Jan. 2017, ISBN: 978-1-78728-583-5.

[9] T. Hunter, Ed., Advanced Microservices: A Hands-on Approach to Microservice Infrastructure and Tooling. Apress, Jun. 2017, ISBN: 978-1484228869.

[10] "LeanIX Microservices Survey," 2017, URL: https://www.cio.de/a/microservices-machen-die-it-schneller-und-agiler,3329517 [accessed: 2017-12-06].

[11] "Microservices trends 2017: Strategies, tools and frameworks," 2017, URL: https://jaxenter.com/microservices-trends-2017-survey-133265.html [accessed: 2017-12-06].

[12] "Number of German passenger cars," 2017, URL: https://www.kba.de/DE/Statistik/Fahrzeuge/Bestand/bestand_node.html [accessed: 2017-12-06].

[13] "Apache Hadoop 2.7.4," 2017, URL: http://hadoop.apache.org/docs/stable/ [accessed: 2017-12-06].

[14] "Spark Overview," 2017, URL: https://spark.apache.org/docs/latest/ [accessed: 2017-12-06].

[15] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," OSDI 2004, vol. Google Inc, 2004, pp. 2,4,10.