Partitioning and Multi-Core Parallelization of Multi-Equation Forecast Models

Lars Dannecker[1], Matthias Boehm[2⋆], Wolfgang Lehner[2], Gregor Hackenbroich[1]

[1] SAP AG, SAP Research Dresden,
Chemnitzer Str. 48, 01187 Dresden, Germany
{lars.dannecker,gregor.hackenbroich}@sap.com
[2] Technische Universität Dresden, Database Technology Group
Nöthnitzer Str. 46, 01187 Dresden, Germany
{matthias.boehm,wolfgang.lehner}@tu-dresden.de

Abstract. Forecasting is an important analysis technique used in many application domains such as electricity management, sales and retail and, traffic predictions. The employed statistical models already provide very accurate predictions, but recent developments in these domains pose new requirements on the calculation speed of the forecast models. Especially, the often used multi-equation models tend to be very complex and their estimation is very time consuming. To still allow the use of these highly accurate forecast models, it is necessary to improve the data processing capabilities of the involved data management systems. For this purpose, we introduce a partitioning approach for multi-equation forecast models that considers the specific data access pattern of these models to optimize the data storage and memory access. With the help of our approach we avoid the redundant reading of unnecessary values and improve the utilization of the CPU cache. Furthermore, we utilize the capabilities of modern multi-core hardware and parallelize the model estimation. Our experimental results on real-world data show speedups of up to 73x for the initial model estimation. Thus, our partitioning and parallelization approach significantly increases the efficiency of multi-equation models.

Key words: Forecasting, Multi-Equation, Partitioning, Parallelization

1 Introduction

Forecasting is used as the basis for decisions in many application areas such as electricity management, sales and retail, and, traffic predictions. Due to recent developments in these domains the employed statistical models face additional challenges and requirements. Typically the available time for estimating the models and providing accurate predictions is significantly decreasing, which requires more efficient data processing capabilities in the employed data management systems. In the energy domain, for example, the emerging smart grid technology and the integration of more renewable energy sources (RES), require real-time

⋆ The author is currently visiting IBM Almaden Research Center, San Jose, CA, USA.

capabilities for balancing the energy demand and supply. Research projects such as MIRABEL [1], and MeRegio [2] address the issues of real-time energy balancing and improved utilization of RES by introducing new developments like dynamic price signals, special energy storage, and demand-response systems. A fundamental prerequisite for current approaches in this area including the balancing of energy in real-time is the availability of accurate forecasts at any time.

Forecasting employs mathematical models—known as forecast models—that model the behavior and development of historic time series. The most important classes of forecast models are autoregressive models [3], exponential smoothing models [4] and models that apply machine learning [5]. Most models use a number of parameters to express specific characteristics of the time series such as seasonal patterns or trends. These parameters are adapted to the specifics of a time series by estimating them on a training data set, with the goal to minimize the forecast error that is measured in terms of an error metric. Typically, each domain exhibits specific characteristics of their time series and thus, employs tailor-made forecast models that address these characteristics. With respect to energy demand time series, e.g., we observe three typical seasonal patterns for the day, the week (working days, weekend) and the year (summer, winter). While our approach can be applied to several application domains, for the remainder of this paper we use the energy domain as a running example.

A model class that is often used for forecasting time series with seasonal behavior comprises multi-equation forecast models [6,7,8,9]. In contrast to typical single-equation models that use just one equation to describe the complete time series behavior, multi-equation models apply an individual sub-model for each specific time period within a selected season, often the daily season (e.g., one model every 30 min). This partitioning of the forecast model allows each sub-model to describe a simpler behavior of the time series, compared to describing all seasonal patterns in one equation. However, the trade-off for using multi-equation forecast models is that now multiple sub-models must be estimated, with each sub-model exhibiting a multi-dimensional search space that exponentially increases with the number of parameters. This leads to higher efforts for the estimation process that typically comprises a large number of iterations per model. In addition, changing time series characteristics caused by continuously available new measurements require the adaptation of forecast models, which typically involves the re-estimation of all model parameters. The re-estimation is almost as expensive as the initial model estimation and thus, very time consuming, especially when using multi-equation models. This clearly contradicts to the requirements given by the real-time balancing.

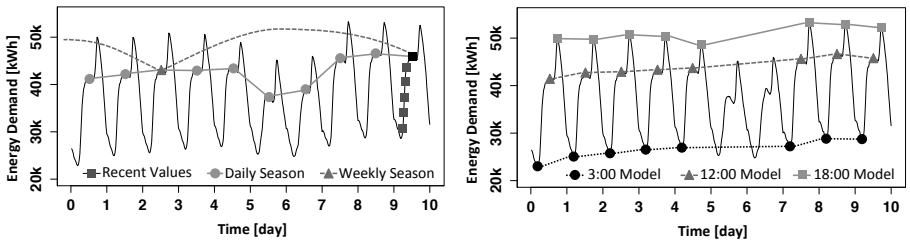
To still allow the use of highly accurate multi-equation models in the face of real-time environments, it is necessary to optimize their calculation efficiency and thus, decrease the time needed for the model adaptation. One direction to overcome that performance bottleneck is to exploit modern hardware architectures. In this context, we observe two major characteristics. First, modern multi-core hardware systems offer a steadily increasing degree of parallelism, since the performance gain from increasing the core frequency is limited by physical con-

straints such as an increasing heat loss and power consumption. Second, with an increasing amount of available main memory, it is possible to store and process all considered data directly within the main memory and thus, to avoid reads from the hard disk. However, access times and bandwidth of the main memory do not increase as fast as the computational power, for what reason memory latency and bandwidth became new limiting factors for the reachable performance [10,11]. Thus, to reduce the influence of the memory latency it is important to optimize the data locality within the main memory and store the data for sequential reading instead of random access. As shown in existing work, the performance of algorithms and software greatly benefits from specifically adapting their data storage and memory access to such hardware characteristics [12]. For this purpose, we present an optimization approach that utilizes the specific time series access pattern of multi-equation forecast models to optimize the data storage with respect to modern hardware. With the help of this framework, we provide for each sub-model only the data it needs for its own calculations and avoid accessing unnecessary information. In addition, our approach optimizes the data locality and cache utilization, which greatly improves the data processing speed. In addition, we utilize the increasing parallelization capabilities of modern multi-core hardware systems and parallelize the parameter estimation of the involved sub-models. This helps us to further speed-up the parameter estimation and to meet the requirements posed by real-time environments. The paper makes the following contributions:

- First, we describe the background of multi-equation models in Section 2.
- Second, we present our partitioning approach that optimizes the data storage with respect to the access pattern of multi-equation models in Section 3.
- Third, we describe parallelization strategies that exploit model inter-similarities to estimate all sub-models in parallel in Section 4.
- Fourth, we present the results of our evaluation that show significant speed-up for the parameter estimation of multi-equation models in Section 5.
- Finally, we present related work in Section 6 and conclude the paper in Section 7.

2 Background of Multi-Equation Forecast Models

There are two typical model classes used for forecasting, namely single-equation and multi-equation models. Single-equation models describe the complete time series behavior including all patterns and seasonalities within one equation. This means that they consider the most recent predecessor values from the time series as the basis for their calculation. In addition, they use further information like seasonal values or external information (e.g., weather) to increase the forecast accuracy. The example presented in Figure 1(a) considers the five most recent values from the time series plus the respective value at the same time one day and one week ago. Popular examples of single-equation models are Box-Jenkins Models (e.g., ARMA, SARIMA) [3] and adaptations of exponential smoothing (e.g., like introduced by Taylor et al. [13]).



(a) Single-Equation Model (b) Multi-Equation Model

Fig. 1. Considered Values of Single-Equation and Multi-Equation Models.

In contrast to single-equation models, multi-equation models avoid the modeling of complex seasonal patterns by decomposing the forecast model and assigning individual sub-models to each time period within a selected season. There, each sub-model is a separate instance of the forecast model equation, with individual values for the comprised parameters. In the energy domain, well known representatives of this model class are the EGRV forecast model [6], first order stationary vector regression from Cottet and Smith [7] and, the PCA based forecasting method from Taylor and McSharry [8]. The reason for splitting up the forecast model with respect to a seasonal pattern is to ease the time series behavior a sub-model has to describe and thus, to increase the forecasting accuracy. The underlying assumption is that successive time series values corresponding to a specific time differ only slightly from season to season. Thus, the current time series value is very similar to previous time series values at the same time. In the energy domain, typically the models are divided with respect to the daily season, leading to the assignment of separate sub-models to each data point within a day. For data in hourly granularity this means that for each hour a specific sub-model is used. Some multi-equation models also consider more than one season in their partitioning. The EGRV model, for example, also considers the weekly season by assigning separate models to weekends and working days in addition to the hourly models. It is important to note that the assignment of individual sub-models to specific time frames limits the use of multi-equation models to time series with equidistant data points. However, this is in line with most forecast models, which generally also require equidistant observations.

When predicting future values, each sub-model calculates a value for the next day that corresponds to its assigned time frame. Thus, to provide a complete one-day-ahead forecast, each sub-model produces exactly one value. Other than single-equation models, the sub-models base their calculation on historic values from their respective time frames; the 8:00 am sub-model, for example, considers historic values that correspond to 8:00 am. However, some additional components like for example lagged error values, might still use values from other time frames. Figure 1(b) illustrates the pattern of the considered values for the sub-models at 3:00 am, 12:00 noon and 6:00 pm. This is also the specific time series access pattern we exploit for a more efficient physical data partitioning.

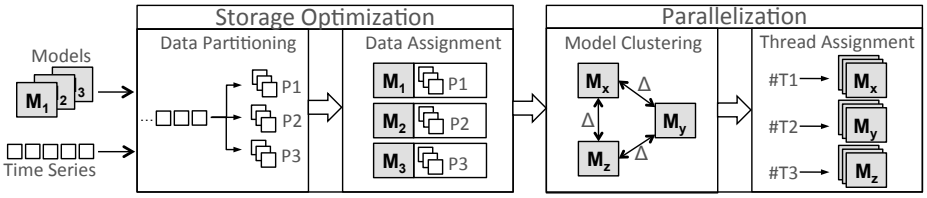


Fig. 2. Multi-Equation Model Optimization Process.

An example sub-model of the EGRV forecast model is shown in Equation 1:

$$\begin{aligned} \text{Hour1} &= \alpha\text{Deterministic} + \beta\text{Temperature} + \gamma\text{Load8} + \delta\text{Lags} \\ \text{Lags} &= \delta_1 y_{t_{-24}} + \delta_2 y_{t_{-48}} + \delta_3 y_{t_{-72}} + \delta_4 y_{t_{-96}} + \delta_5 y_{t_{-120}}. \end{aligned} \tag{1}$$

There, the *Deterministic* variables represent additional calendar information and are included as dummy variables (value 0 or 1). Typically they do not require to read time series values. Variables from the *Temperature* category use values from an external temperature time series. In this paper, we do not separately describe the handling of this external information, since our partitioning approach works analogous for such time series. The variables named as *Load8* represent a specific aspect of the EGRV model and correspond to the load at 8:00 am on the previous day. Finally, the *Lags* variables represent the last five time series values that correspond to the specific time of each sub-model (e.g., $t_{x-24}, t_{x-48}, t_{x-72}$, etc.).

3 Partitioning for Multi-Equation Forecast Models

The core idea underlying our approach is to physically partition the time series in a way that reflects the model partitioning of the multi-equation model. For this purpose we employ the process illustrated in Figure 2. There, we first partition the data and assign each data partition to its corresponding sub-model. Therefore, we ensure that each model physically accesses only the portions of the time series it needs for its own calculations. This avoids the constant scanning of unnecessary additional values and thus, significantly increases the calculation speed of the complete model. In addition, modern multi-core hardware systems offer an increasing amount of parallelism that we exploit by estimating the involved sub-models in parallel. This is done in the second part of the process. As we assume a larger number of models compared to the number of available threads, we also optimize the thread assignment of these models. The idea is to use similarities between the sub-models and exploit the parameters estimated for one model as the input for the estimation of the successive model. The goal is to reduce the number of iterations until the optimization algorithm converges. As a last step we execute the parallel parameter estimation for all sub-models.

The estimation of forecast model parameters is typically conducted using local (e.g., gradient descent, L-BFGS-B) or global search algorithms (e.g., Simulated Annealing). This optimization task involves a large number of iterations,

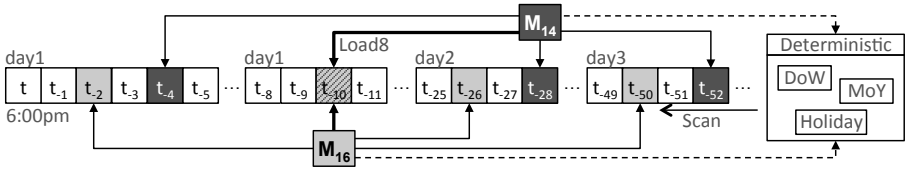


Fig. 3. Initial Sub-Model Time Series Access.

where each iteration requires to read all necessary time series values. With respect to single-equation models this means that the complete time series is scanned to evaluate the error of the chosen parameter combination. For multi-equation models different parameters are assigned to each sub-model and thus, each model is estimated separately. For each sub-model estimation only the values corresponding to the assigned time frame plus potentially some commonly used variables are required. Figure 3 illustrates the initial situation using the EGRV model. There, the sub-models M14 (corresponds to 2:00 pm) and M16 (corresponds to 4:00 pm) are presented. Both models only require their specific time series values, namely t_{-4}, t_{-28}, t_{-52} for M14 and t_{-2}, t_{-26}, t_{-50} for M16. In addition, both consider the time series values corresponding to the Load8 variable and some deterministic variables (that do not require time series access).

The issue in the non-partitioned case is that the time series is stored chronologically in a single, large array and reading values from cache or memory always requires to read a full cache line or—depending on the hardware platform—even larger block granularities. The number of values contained in a cache line depends on the system specification. Using the Intel Core i7, for example, a cache line contains 64Byte and a double value is 8byte, which results in 8 time series values provided per cache line. Hence, each cache line read for a required value, will also contain time series values that do not correspond to a sub-model's time frame and thus, are not needed for the sub-model estimation. In particular, each cache line will contain only a single required value. Model 16 in Figure 3 for example only needs the value t_{-26}, but the read cache line will also provide the values t_{-25},t_{-27},t_{-28}, etc. As a result, multi-equation models have a specific time series access pattern that does not correspond to the time series storage and hardware access pattern. Thus, the number of read time series values and cache lines increases with the number of sub-models, where the majority of the read values are not required for a sub-model estimation. This results in a large overhead and a poor data locality within cache and main memory, which leads to long estimation times when working with multi-equation models.

To allow multi-equation models to quickly adapt to new situations and thus, to better meet the requirements posed by real-time applications, we optimize the time series storage to reduce the number of unnecessary time series reads. To do so, we partition the time series in a way that corresponds to the time series access pattern of multi-equation models. The number of partitions directly matches the number of involved sub-models, which in most cases also reflects the

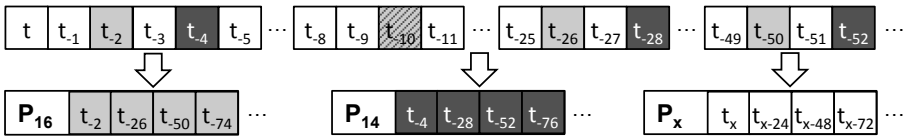


Fig. 4. Time Series Partitioning.

granularity of the time series. Each of the used partitions represents a specific time frame (e.g., 4:00 pm) and exclusively stores only values that pertain to this time frame. Since multi-equation models only support equidistant time series (compare Section 2), each partition comprises an equal number of values. Figure 4 illustrates an example partitioning for the EGRV model. There, Partition 14 contains only the values that correspond to 2:00 pm and Partition 16 only the values that belong to 4:00 pm. In addition, we replicate the values that are commonly accessed by all sub-models (e.g., t_{-10} - Load 8) and store the replicates in the partitions as well. This ensures the independence of the models for an optimal further parallelization. After the partitioning is finished, the partitions are assigned to the corresponding sub-model, i.e., the sub-model that describes the same hour the values from a partition belong to.

Each created partition persists in a specific area of the main memory, which means that values belonging to the same partition, i.e., values that are assigned to the same sub-model, are stored closely together within the same memory area. As a result, when reading the time series values, instead of jumping from value to value and reading cache lines that contain unnecessary values, the sub-models can sequentially process all values stored within their respective partition. This sequential reading of time series values directly increases the processing performance. In addition, the tight data storage, results in more necessary values that are contained in a single cache line. As a result, the number of cache lines and memory pages read during a sub-model estimation decreases and thus, the number of cache and memory accesses. Furthermore, for currently running estimations more necessary values can be stored in the different cache levels. This greatly increases the processing speed of the CPU and decreases the number of cache misses. Figure 5 compares the data storage within the cache for the non-partitioned and partitioned case of sub-model M16 - 4:00 pm. There, again we refer to the Intel Core i7 CPU, where each cache line contains 8 time se-

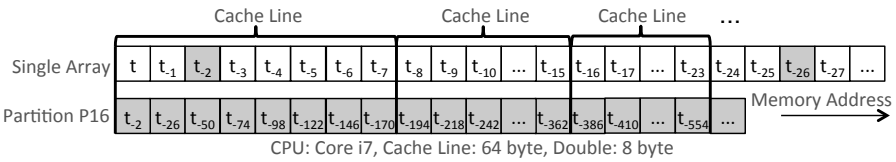


Fig. 5. Cache Organization of Non-Partitioned and Partitioned Time Series.

ries values. In the non-partitioned case this means that only one of the values contained in a cache line is needed for the sub-model estimation. In contrast, when using our partitioning approach, all values in a cache line are necessary for the estimation process. Thus, our partitioning approach reduces the number of cache lines processed by the CPU and therefore, reduces the calculation time for each iteration of the estimation algorithm. As a result, the memory and cache locality leads to more efficient calculations for estimating the parameters of all sub-models. Furthermore, the cache locality and the sequential storage of values within a partition also enable the usage of compression techniques like the patched frame of reference (PFOR). Hence, even more values could be stored within a single cache line, which could further increase the processing speed. In the future we will intensively evaluate the use of compression technologies in conjunction to our time series partitioning.

Altogether, our partitioning approach ensures that during all iterations of the optimization algorithm each sub-model only reads its necessary time series values and therefore, reduces the amount of redundantly accessed additional values. This also leads to an optimized memory locality of the data and thus, reduces the number of read cache lines and the amount of cache misses. This greatly improves the calculation performance of the optimization algorithms and greatly decreases the time needed for the multi-equation model parameter estimation.

4 Parallelization of Independent Forecast Models

We further optimize the multi-equation model estimation, by exploiting the parallelization capabilities of modern multi-core hardware. Therefore, we assign all sub-models including their respective partitions to a number of threads that execute the parameter estimation of the sub-models in parallel. Due to the fact that memory throughput and latency can quickly become the limiting factors when using multi-core parallelization, the parallel estimation also profits from the optimized storage and enhanced cache utilization provided by our partitioning approach. Ideally, the number of utilized threads would exactly match the number of involved sub-models, which would also bring the greatest benefit for the parallelization. However, in the real world the number of threads that can be directly executed in parallel on a specific system is limited. The number of these so called hardware threads is typically much smaller compared to the number of involved sub-models (e.g., 48, 96). A system employing a quad-core Intel Core i7 CPU, for example, has eight threads available; one hardware thread and one additional thread using Hyper-threading can be executed per core. Assigning more threads than available hardware threads creates no additional benefit in our scenario, but rather induces additional costs due to overhead for thread scheduling and cache displacement issues. As a result, for the parallel execution of the sub-model estimation, we limit the number of parallel threads to the number of hardware threads available on the executing hardware system and thus, assign multiple sub-models to each thread for sequential estimation. The assignment of the sub-models to the threads is typically conducted using a task queue,

Table 1. Test Results: Parameter Equality of Three Example Models.

	P1	P7	P8	P9	P12	P15	P16	P20
M11	0.5250	0.9991	0.9883	0.9990	0.3798	0.4762	0.3710	0.5355
M16	0.5511	0.9974	0.9633	0.8651	0.3445	0.3161	0.3530	0.5441
M17	0.9989	0.9078	0.9928	0.8645	0.3409	0.3857	0.3012	0.5523

where each thread picks the next sub-model as soon as it finished the previous parameter estimation. This leads to good load balance, even if time for estimating sub-models differs significantly. However, on average when assuming, e.g, 48 sub-models and 8 hardware threads, each thread estimates six sub-models.

Given the thread-local serial estimation of a subset of models, we want to further optimize the sequential estimation for the sub-models assigned to one thread. Due to the fact that some sub-models describe similar shapes, we assume that these models should also have similar parameter combinations. In a small experiment we compared the parameters between sub-models after their initial estimation and the results supported our assumption for the most part. Table 1 presents some example parameters for three example models M11, M16 and M17. While the assumption holds for most parameters, it does not for all. Some parameters still differ for sub-models we identified as similar (marked grey in Table 1). In our example, this concerns P1 and P7 for model M17 as well as P9 and P15 for model M11. Still, both models keep their relative similarity to model M16. As a result, the parameter combinations of similar models are a much better approximation of a good starting point for the parameter estimation, compared to starting from the origin. Thus, the basic idea of our improvement is to iteratively provide the result of the preceding parameter estimation, as the input (i.e., the starting value) for the estimation of the subsequent sub-model. This *sequential start* approach reduces the number of necessary iterations for the subsequent optimization algorithms per thread, because only the first sub-model in each thread needs the full effort for the parameter estimation. All subsequent models then profit from better suited starting parameters, for what reason this should greatly reduce the time needed for the parameter estimation.

For the parameter re-estimation we can even go one step further, because the parameter values of the sub-models were already determined in the initial estimation. As described above, some sub-models exhibit a large portion of very similar parameter values. For this reason, we enhance the sequential start approach by clustering the most similar sub-models and assigning each cluster to a single thread. To do so, we measure the distances between the individual values of the parameters for all sub-models (e.g., $\text{dist}(\alpha_{M1},\alpha_{M2})$, $\text{dist}(\alpha_{M1},\alpha_{M3})$) using the *euclidean distance measure* and combine the models with the least distance to each other into one cluster. The number of used clusters directly corresponds to the number of involved threads. In detail, we follow the *k-means clustering* approach using the following process:

1. Sub-models are estimated. Maximum model number per thread calculated.
2. Sub-models are randomly assigned as centroids.

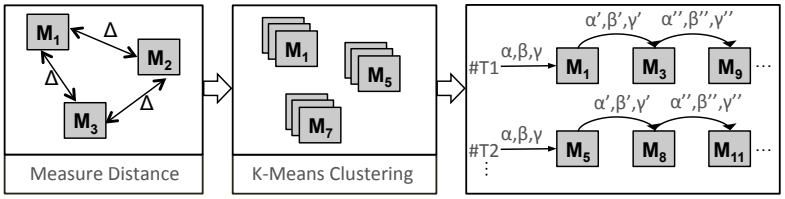


Fig. 6. Clustered Parallelization with Sequential Start.

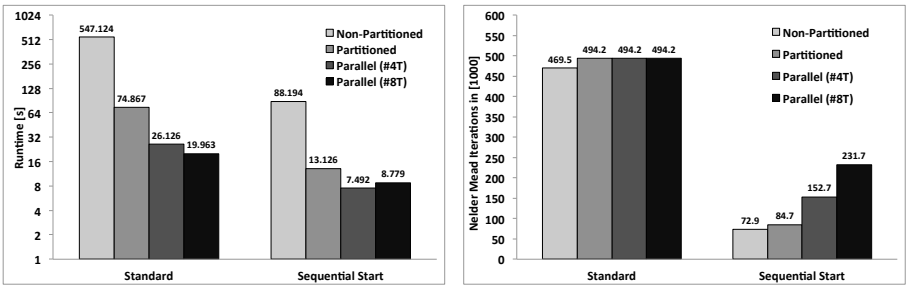
3. Distance between centroids and sub-models is computed.
4. Models are assigned to centroid with minimal distance unless thread is full.
5. If thread is full, model is assigned to next best thread.
6. The following steps are repeated until no sub-model changes thread anymore.
   (a) Incrementally compute new centroid from all sub-models per thread.
   (b) Measure distance between new centroids and all sub-models.
   (c) Reorder models with respect to new distance measures.

As soon as our clustering process is finished, we assign the clusters to the respective threads and execute sequential start parameter estimation for each thread. To avoid degeneration in the sense that one thread estimates much more models than the other ones, we place a constraint that all threads execute the same number of models if possible. Figure 6 illustrates the parallelization process. Due to the stronger similarity between the sub-models within one thread, we can even further reduce the number of iterations conducted by the parameter estimation algorithms and thus, reduce the time needed to estimate all sub-models.

To sum up, our parallelization approach further increases the efficiency of multi-equation models. To compensate for the limited number of hardware threads on most systems, we exploited sub-model inter-similarities to optimize the parallelization. While we described our parallelization approach for a local multi-core system, it is also possible to apply the approach to a distributed setting. There, the partitioning approach would be of even more value, because it would limit the amount of transmitted data between the involved systems.

5 Experimental Evaluation

In this evaluation, we substantiate the claims of our multi-equation model optimization approach and show that with the help of our partitioning and parallelization we can greatly reduce the time needed for estimating multi-equation forecast models. Our evaluation compares the time needed for estimating and re-estimating all sub-models, the number of iterations necessary for the (re-)estimation, the amount of cache misses and, the scalability of our approach. For this purpose, we employed the EGRV forecast model as introduced in Section 2). For the parameter estimation we used the Nelder Mead Downhill Simplex approach [14] as a local optimization algorithm. The employed dataset is the energy demand data from the UK National Grid: *National Grid Demand* (publicly



(a) Runtime per Optimization (b) Iterations per Optimization

Fig. 7. Different Optimizations for Parameter Estimation.

Table 2. Average Runtime per 1000 Iterations for used optimization approaches.

Variant	Non-Partitioned	Partitioned	Parallel (#4T)	Parallel (#8T)
Avg. Runtime	1.2245s	0.1566s	0.0521s	0.0393s

available [15]): Electricity demand of the United Kingdom. Measures: INDO, January 1st 2002 to December 31st 2009, 30min resolution (140256 values).

For our evaluation we used the following test system: Quad-Core Intel Core i7 2635QM (2.0 GHz), 4GB RAM, 128GB SSD, Mac OSX 10.6.8. Our forecasting test suite is written in C++ using the GCC 4.2.1, with OpenMP for the parallelization. We configured OpenMP to use a pre-defined, static number of threads and dynamic thread assignment. Thus, if not explicitly specified (like for the clustering) upon finishing its job, each thread estimates the next sub-model in the queue. All presented results are the average of 20 subsequent runs.

5.1 Parameter Estimation

In the first experiment, we evaluated the runtime necessary for the estimation of a multi-equation model, using our optimization techniques. Thus, we compared the standard, non-partitioned case with the partitioned and parallelized versions. In addition, we also compared the use of the Sequential Start method. Figure 7 illustrates the results. Please note the logarithmic scale that is used in the graphs. There, in Subfigure 7(a) the most important fact is that solely the partitioning approach reduced the time needed for the estimation of all sub-models from 547.124s to 74.867s, even with slightly more iterations (compare Subfigure 7(b)). This is a significant improvement over the non-partitioned case. The parallelization then further reduces the necessary runtime. It can be seen that the runtime improvement is much larger from 1 thread to 4 threads compared to from 4 threads to 8 threads. The reason is that when using 4 threads each of them can be directly executed on one core, while when using 8 threads the additional 4 threads are subject to Hyper-threading. The sequential start method further decreases the necessary runtime for all variants. The runtime

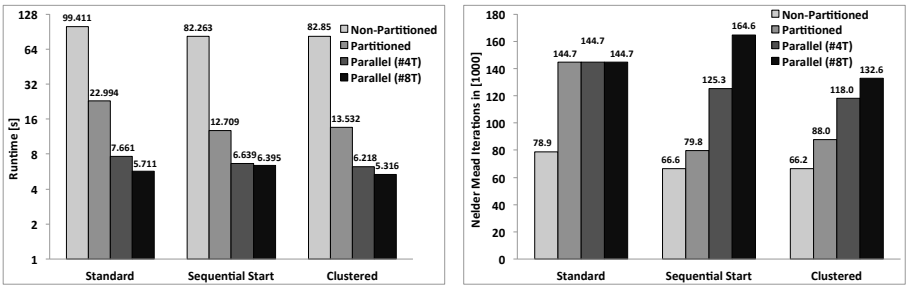
of the non-partitioned version is reduced from 547.124s to 88.194s (factor 6.2) and the runtime of the partitioned variant is reduced from 74.867s to 13.126s (factor 5.7). The reason for the reduced runtime is illustrated in Figure 7(b). There, we can see the reduction of the number of iterations conducted until the optimization algorithm converges. Thus, the sequential start provides more suitable starting points for the parameter estimation then starting from the origin for each sub-model. However, it is important to note that the benefit of the sequential start method depends on the number of forecast model parameters and the used estimator. We can further see that the number of iterations for the parallelized methods increases with the number of threads. The reason is that a full estimation is necessary for the first model that is estimated per thread; e.g., for 8 threads, 8 models cannot exploit the parameters from previous models. In addition, when using a larger number of threads, fewer models are estimated by one thread sequentially. Thus, the chance for having only non-similar sub-models assigned to a single thread is higher compared to using 4 threads, where more models are estimated in a row. As a result, the parallelization with 8 threads needs more time using the sequential start optimization then the parallelization with 4 threads. This means that the benefit of the hyper-threaded 4 additional threads is of less value, then the drawback due to the increased number of iterations. This leads us to the problem of automatically assigning an optimal degree of parallelism that we will address in the future.

Table 2 presents the average runtime for all optimizations per 1000 iterations (chosen for better readability). The sequential start and clustering approaches are not listed, because they do not influence the runtime per iteration. The results show a clear trend for the optimization approaches. While the non-partitioned variant needs more than 1.2 seconds for 1000 iterations of the Nelder Mead algorithm, all other approaches are clearly below one second, with the partitioned version marking the maximum of the optimization approaches with 0.1566s.

Overall, we can see that with the help of our optimizations the parameter estimation can be conducted in a few seconds compared to minutes needed for the non-partitioned method. Especially the partitioning optimization provides a significant speed-up. Thus, our optimizations make sure that multi-equation models can be used in the face of the challenged posed by the market dynamics.

5.2 Parameter Re-Estimation

Our second experiment is similar to the first one, but we compare the runtime necessary for the parameter re-estimation rather than for the initial estimation. Thus, parameters of the sub-models are not estimated from scratch, but the local search algorithms can start from the last valid parameter combination. Typically the parameter re-estimation is triggered after additional values were added to the time series. Thus, we appended 1440 additional values (i.e., one month) to the time series used for the initial parameter estimation and triggered the re-estimation afterwards. The results are illustrated in Figure 8. There, the results of the standard execution method in Figure 8(a) are similar to the results of the initial estimation. The partitioning greatly reduces the necessary



(a) Runtime per Optimization (b) Iterations per Optimization

Fig. 8. Different Optimizations for Parameter Re-Estimation.

runtime, while the parallelization distributes the models to multiple threads and thus, also speeds up the re-estimation. The sequential start reduces the time necessary for the re-estimation, but the decrease is not as significant as for the initial estimation. The reason is that the previous parameters are already a good approximation of a starting point for the local optimization algorithm. For the parallelization with 8 threads, the usage of the sequential start method even increases the necessary runtime. The reason is similar to the causes presented for the initial estimation, but additionally in some cases the parameters from the previous sub-model are worse starting points compared to the old parameters.

For the parameter re-estimation we also used our clustering method described in Section 4. There, the models are not sequentially assigned to the threads, but according to the clustering result. The clustering clearly improves the results of the parallelized version, because it provides better starting points and a more beneficial thread assignment of the sub-models than the sequential start method. Also, using our clustering approach, the calculated centroid is provided as starting point to the first models, which turns out is also a better approximation of a good start than the old parameters. Figure 8 again illustrates the results. There, the decrease/increase of the iterations correspond to the measured runtimes. Especially the increased number of iterations when using the sequential start in conjunction with the 8-thread parallelization is interesting, because it supports the assumption that sequential model assignment is worse compared to using just the old parameter combinations. The runtimes for the single-threaded, partitioned and non-partitioned versions stay roughly the same. The only slight increase is reasoned by the k-means overhead that is still conducted even for those variants. Also, increasing the number of clusters to 4 and 8 for the single threaded versions brings no benefit. The reason is similar to the simple sequential start method, meaning that an increasing number of clusters also increases the number of models that cannot benefit from the sequential start.

Overall, the re-estimation exhibits similar results like the initial estimation. However, the overall runtime for the re-estimation is lower, due to better starting points for the local search algorithm. With the help of our optimizations we also

reduced the necessary runtime for adapting a multi-equation model significantly in all cases. The runtime stays always just roughly over 10s for the partitioning approach and with respect to the parallelization the time further decreases.

5.3 Cache Utilization

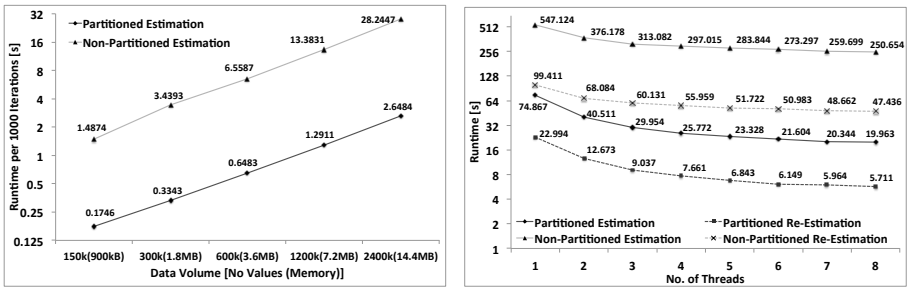
In this experiment we compared the cache misses for the partitioned, non-partitioned and parallelized case. For this purpose, we used the Intel Performance Counter Monitor that evaluates the values of the Performance Management Unit located directly on modern Intel CPUs. With the help of this tool we measured the number of cache misses that occurred in 20 seconds while running the estimation of an EGRV model. The used Intel Core i7-2635QM with 4 cores provides a 6 MB L3 cache and 256kB L2 cache per core. Our test data set contained 122,736 values, which resulted in a size of 737kB. This means that the complete test data set can be cached in the L3 cache and thus, we expect high L3 cache hit rates in all cases. The results are presented in Table 3. As expected, all cases exhibit a very high L3 cache hit rate. However, the non-partitioned case exhibits a far higher total number of cache misses. Due to the fact that the cache hit rate is nevertheless comparable, this means that in the non-partitioned case far more reads were executed on the L3 cache. This supports the assumption that in the partitioned case less cache lines must be read from the L3 cache. With regard to the L2 cache the result is more diverse. There we see a very low L2 hit rate of only 2% for the non-partitioned case, in comparison to almost 100% for both partitioned cases. This means that storing the data tight together leads to far more necessary values in the cache and thus to a low number of cache misses. The higher total number of L2 cache misses for the parallelized case is reasoned by the execution on 4 threads. The Intel Core i7 has one L2 cache for each CPU core and thus, the number of cache misses roughly increases with the number of threads. As a result, our partitioning approach greatly optimizes the cache utilization, which results in a very high L2 cache hit rate.

5.4 Scalability

In the last experiment we evaluate the scalability of our approach regarding data volume and number of threads. The results are presented in Figure 9. Please note the logarithmic scale of the y-axis. We first compared the behavior of our algorithm with an increasing data volume. For this evaluation we used synthetic data sets with different sizes. The drawback of synthetic data sets is that the

Table 3. Test Results: Cache Misses per Storage Approach.

	# L3 Misses	% L3 Hits	# L2 Misses	% L2 Hits
Non-Partitioned	206.850 Mio.	93%	8,034.0 Mio.	2%
Partitioned	237,000	99%	7.245 Mio.	96%
Partitioned (4T)	143,000	100%	22.714 Mio.	97%



(a) Increasing Data Volume (b) Increasing Number of Threads

Fig. 9. Scalability of Our Optimization Framework.

number of iterations varies when changing the data volume, which distorts the results. Thus, the runtime is for 1000 iterations to eliminate the dependency on the number of iterations. In Figure 9(a)), we see a linear increase of the necessary runtime time for the partitioned and non-partitioned estimation. Meaning that doubling the data volume also doubles the runtime of the parameter estimation. In addition, the development of our partitioning approach is constantly below the non-partitioned case and both have a similar pace of increase. Thus, there won't be a data volume where the non-partitioned case is faster than the partitioned case, which clearly renders the advantages of the time series partitioning.

The results for the scalability concerning an increased number of threads are presented in Figure 9(b). Concerning all cases, we observe the greatest runtime decrease for the parameter estimation and re-estimation, when increasing the number of threads from one to two. When adding more threads, the runtime benefit decreases. Furthermore, the start of the Hyper-threading clearly marks a specific point, after this point the runtime gain is only marginal. In the case of the used Intel Core i7, we have four cores available, meaning that the largest benefit can be observed up to a number of 4 threads. When using more than four threads the additional benefit decreases significantly. As a result, the number of used threads should at least match the number of available cores on a system, because the additional performance gain from hyper-threading is limited. Moreover, we can see that the performance gain when increasing the number of threads is higher for our partitioned case. When increasing the number of threads from one to two, the runtime decreases for the partitioned case by almost one half and for the non-partitioned case by only one third. This trend continues when further increasing the number of threads (e.g., 2 -> 3 Threads: 26% partitioned, 17% non-partitioned). Finally, the runtime difference between one and eight threads is also higher for our partitioned case. The runtime decreases by almost 75% for the partitioned case, whereas for the non-partitioned case the runtime only decreases by around 50%. As a result, our partitioning and the optimized cache utilization also increased the possible degree of parallelization and thus, the performance gain from the parallelization is significantly higher.

6 Related Work

Current research mostly focuses on increasing the accuracy of forecast models. Therefore, there exists only few related work regarding the partitioning and parallelization of forecast models, in particular multi-equation models. Some multi-equation models such as those introduced by Cottet and Smith [7], Soares and Medeiros [9], and Taylor et al. [8] directly include performance optimizations on the logical level. Taylor, for example, proposes to use Principal Component Analysis in conjunction with his multi-equation model to reduce the number of sub-models and thus, the time needed for model estimation. However, the introduced approaches directly modify the model calculations and thus, influence the achievable accuracy. In contrast, our optimizations only change the memory access of the models and do not change the model's calculations, which means that the resulting accuracy is not influenced. In addition, our proposed optimizations on the physical level can be used together with the optimizations on the logical level and thus, increase the performance of these approaches even further.

However, there is also some work that—like our approach—uses optimizations on the physical level. Ge and Zedonik [16] propose to use a skip-list with various levels to provide different data granularities and different history length for various purposes and forecast horizons. Forecasts with a long horizon (> 1 month) use a very coarse grain granularity, while forecasts with a short horizon use very fine-grained data. A similar approach is presented by Agrawal et al. [17] where they merge similar attributes into subset and calculate forecasts only on these subsets. These approaches can greatly reduce the amount of time needed for the estimation of a forecast model. However, reducing the amount of data in most cases also reduces the reachable accuracy of a forecast model. Especially in application domains, where complex patterns are described, the use of fine-grained data and a suitable history length is required for providing accurate forecasts. Our approach does not reduce the number of beneficial values considered during model estimation, but reduces the amount of redundantly read unnecessary values. This means that the accuracy is not influenced by the optimizations proposed by our approach. Also some approaches that directly involve data partitioning and parallelization of forecast models exist. Canas et al. proposed a partitioning solution for neural networks that forecast river-flows to speed up the calculations [18]. Kalaitzakis et al. propose a parallel neural network for forecasting electric load [19]. While the first approach partitions the values into river-flow-specific categories that are provided separately to the neural network, the second approach proposes a parallel calculation of hourly load, similar to multi-equation models. Overall, their partitioning is rather a model decomposition than an optimization on the physical level, especially because the publications omit details about the storage structure and parallelization. In addition, the proposed parallelization only provides the naive approach for calculating the available neurons of the neural network in parallel. In contrast, a massive parallelization approach is provided by Shimokawabe et al. [20]. They propose to distribute the calculation of a weather forecast model on a super computer that provides some thousand GPUs. However, the considered forecast

model is a physical weather model, which means that their solution is specific to their approach and cannot directly be applied to multi-equation models.

Overall, our approach is the first optimization approach that exploits the specific access pattern of multi-equation models to optimize the data storage and cache utilization for less redundant and faster data processing. With the help of this technique we speed up the parameter estimation process significantly and our improvement is greatly above the current state of the art. In contrast to most related solutions, our approach does not change the calculation specifics of a forecast model and thus, does not influence the reachable accuracy. Furthermore, our approach can supplement the presented solutions and as a result, even further increase their efficiency enhancements. In addition, the proposed partitioning and parallelization approach is not limited to models from a specific application domain, but can be applied to all kinds of multi-equation forecast models.

7 Conclusion

In this paper, we presented an optimization approach for multi-equation models that greatly increases the efficiency of such models. Our time series partitioning approach ensures that each sub-model only accesses time series values that are necessary for their specific calculations, which leads to an optimized memory locality of the stored time series values. This greatly reduces the number of cache misses, read cache lines and thus, results in an increased data processing efficiency. We further increase the speed of the parameter estimation, by utilizing modern multi-core hardware systems and estimating the sub-models in parallel. There, we addressed the issue of a limited amount of threads, by presenting our sequential start execution and the clustered thread assignment technique. In our evaluation we showed that our optimization framework significantly reduces the time necessary for estimating and re-estimating an multi-equation forecast model. Especially our partitioning approach achieved a major speed up of the optimization calculations. As a result, with the help of our approaches multi-equation models can be used in the face of real-time environments.

In the future we want to enhance our approach by using compression technologies that further increase the number of values read per cache line. In addition, we want to automatically decide for which purpose to use available parallelism most beneficially. The reason is that it is either possible to use available parallelism to (1) increase the number of models estimated in parallel and thus, to potentially increase the efficiency or (2) to simultaneously start the estimation from different starting points and thus, to potentially increase the accuracy.

Acknowledgment

The work presented in this paper has been carried out in the MIRACLE project funded by the EU under the grant agreement number 248195.

References

1. MIRABEL Project. (2011) http://www.mirabel-project.eu.
2. MeRegio Project. (2011) http://www.meregio.de/en/.
3. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time Series Analysis: Forecasting and Control. John Wiley & Sons Inc. (1970)
4. Winters, P.R.: Forecasting sales by exponentially weighted moving averages. Management Science **April** (1960) 324–342
5. Bunnoon, P., Chalermyanont, K., Limsakul, C.: A computing model of artificial intelligent approaches to mid-term load forecasting: a state-of-the-art- survey for the researcher. Int. Journal of Engineering and Technology **2**(1) (2010) 94–100
6. Ramanathan, R., Engle, R., Granger, C.W., Vahid-Araghi, F., Brace, C.: Short-run forecasts of electricity loads and peaks. International Journal of Forecasting **13**(2) (1997) 161–174
7. Cottet, R., Smith, M.: Bayesian modeling and forecasting of intraday electricity load. Journal of the American Statistical Association **98** (2003) 839–849
8. Taylor, J.W., de Menezes, L.M., McSharry, P.E.: A comparison of univariate methods for forecasting electricity demand up to a day ahead. International Journal of Forecasting **22** (2006) 1–16
9. Soares, L.J., Medeiros, M.C.: Modeling and forecasting short-term electricity load: A comparison of methods with an application to brazilian data. International Journal of Forecasting **24**(4) (2008) 630 – 644
10. Wulf, W.A., McKEE, S.A.: Hitting the memory wall: Implications of the obvious. Computer Architecture News **23**(1) (1995) 20–24
11. Borkar, S.Y., Mulder, H., Dubey, P., Pawlowski, S.S., Kahn, K.C., Rattner, J.R., Kuck, D.J.: Platform 2015: Intel processor and platform evolution for the next decade. Technical report, Intel Corporation (2005)
12. Kim, C., Chhugani, J., Satish, N., Sedlar, E., Nguyen, A.D., Kaldewey, T., Lee, V.W., Brandt, S.A., Dubey, P.: Fast: Fast architecture sensitive tree search on modern cpus and gpus. In: Proceeding of the SIGMOD 2010. (2010)
13. Taylor, J.W.: Triple seasonal methods for short-term electricity demand forecasting. European Journal of Operational Research **204** (2009) 139–152
14. Nelder, J., Mead, R.: A simplex method for function minimization. The Computer Journal **7**(4) (1965) 308–313
15. Nationalgrid UK: Metered half-hourly electricity demands. (2010) http://www.nationalgrid.com/uk/Electricity/Data/Demand+Data/.
16. Ge, T., Zdonik, S.: A skip-list approach for efficiently processing forecasting queries. In: Proceeding of the VLDB 2008. (2008)
17. Agrawal, D., Chen, D., ji Lin, L., Shanmugasundaram, J., Vee, E.: Forecasting high-dimensional data. In: Proceeding of the SIGMOD 2010. (2010)
18. Cannas, B., Fanni, A., See, L., Sias, G.: Data preprocessing for river flow forecasting using neural networks: Wavelet transforms and data partitioning. Physics and Chemistry of the Earth **31**(18) (2006) 1164–1171
19. Kalaitzakis, K., Stavrakakis, G., Anagnostakis, E.: Short-term load forecasting based on artificial neural networks parallel implementation. Electric Power Systems Research **63** (2002) 185–196
20. Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukada, A., Maruyama, N., , Matsuoka, S.: An 80-fold speedup, 15.0 tflops full gpu acceleration of non-hydrostatic weather model asuca production code. In: Proceedings of Super Computing 2010. (2010)