Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

DANIEL REHFELDT, HANNES HOBBIE, DAVID SCHÖNHEIT, AMBROS GLEIXNER, THORSTEN KOCH, DOMINIK MÖST

# A massively parallel interior-point solver for linear energy system models with block structure

ZIB Report 19-41 (revised, October 2019)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30-84185-0
Telefax: +49 30-84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

# A massively parallel interior-point solver for linear energy system models with block structure

Daniel Rehfeldt, Hannes Hobbie, David Schönheit, Ambros Gleixner, Thorsten Koch, and Dominik Möst

Zuse Institute Berlin, Department of Mathematical Optimization,
{rehfeldt,gleixner,koch}@zib.de

Technische Universität Dresden, Lehrstuhl für Energiewirtschaft,
{hannes.hobbie,david.schoenheit,dominik.moest}@tu-dresden.de

October 20, 2019

**Abstract**

Linear energy system models are often a crucial component of system design and operations, as well as energy policy consulting. Such models can lead to large-scale linear programs, which can be intractable even for state-of-the-art commercial solvers—already the available memory on a desktop machine might not be sufficient. Against this backdrop, this article introduces an interior-point solver that exploits common structures of linear energy system models to efficiently run in parallel on distributed-memory systems. The solver is designed for linear programs with doubly-bordered block-diagonal constraint matrix and makes use of a Schur complement based decomposition. Special effort has been put into handling large numbers of linking constraints and variables as commonly observed in energy system models. In order to handle this strong linkage, a distributed preconditioning of the Schur complement is used. In addition, the solver features a number of more generic techniques such as parallel matrix scaling and structure-preserving presolving. The implementation is based on the existing parallel interior-point solver PIPS-IPM. We evaluate the computational performance on energy system models with up to 700 million non-zero entries in the constraint matrix—and with more than 200 million columns and 250 million rows. This article mainly concentrates on the energy system model ELMOD, which is a linear optimization model representing the European electricity markets by the use of a nodal pricing market clearing. It has been widely applied in the literature on energy system analyses during the recent years. However, it will be demonstrated that the new solver is also applicable to other energy system models.

# 1 Introduction

Currently 80 % of the world's energy supply is produced by fossil fuels and there is no doubt this number has to change if we want to limit the amount of $CO_2$ emitted into the atmosphere. Therefore, many countries have started an energy transition, switching towards renewable based energy sources, in particular wind and solar for electricity generation. With decentralizing power generation, electricity networks gain in importance, not only to connect generation with demand locations, but also to provide for necessary flexibility for renewable integration. Consequently, the transition towards a renewable based electricity system comes with high amounts of electricity grid expansion between single markets, but also within a country. In particular, the European network topology can be characterized as highly meshed with more than 3000 transmission grid nodes. Managing such electricity grids undergoes highly complex planning and operation procedures.

Generally, energy system models try to capture, predict, and optimize country or even continental wide energy systems in order to ensure security of supply while minimizing transition costs. It comes to no surprise that these models are huge, as they have to capture spatial, temporal, and technological dimensions. Today, many real-world energy system models (ESMs) are modeled as linear programs (LPs).

There has been tremendous progress in general LP solvers during the last 30 years [4, 15]. Many LPs that were considered intractable two or three decades ago can now be solved within seconds. Still, for large-scale LPs with hundreds of millions of variables and constraints such as those arising from high-resolution energy system models, even the best commercial solvers can take prohibitively long to find an optimal solution. Moreover, such large-scale problems might not even fit into the main memory of a modern desktop machine. One the other hand, the increasing availability of distributed-memory parallel computers offers a huge potential both for reducing solution time and avoiding memory bottlenecks. Unfortunately, however, general state-of-the-art LP solvers cannot (efficiently) run on distributed-memory systems. Therefore, one way forward is to develop more specialized algorithms that are able to exploit the structure of a given problem class for its distributed parallel solution. A typical structure observed in linear energy system model is the so-called *arrowhead* or *doubly bordered block-diagonal* form:

$$
\begin{aligned}
\min \quad & \sum_{i=0}^{N} c_i^T x_i \\
\text{s.t.} \quad & T_0 x_0 & & & & & & & = h_0 & (1)\\
& T_1 x_0 & + & W_1 x_1 & & & & & = h_1 & (2)\\
& T_2 x_0 & + & & W_2 x_2 & & & & = h_2 & (3)\\
& \vdots & & & & \ddots & & & \vdots & \\
& T_N x_0 & + & & & & & W_N x_N & = h_N & (4)\\
& U_0 x_0 & + & U_1 x_1 & + & U_2 x_2 & \cdots & U_N x_N & = h_{N+1} & (5)\\
& x_0, & & x_1, & & x_2, & \cdots & x_N & \geq 0 & (6)
\end{aligned}
$$

where $c_i, x_i \in \mathbb{R}^{n_i}, T_i \in \mathbb{R}^{m_i \times n_0}, U_i \in \mathbb{R}^{m_{N+1} \times n_i}$ for $i \in \{0, 1, ..., N\}$ and $W_i \in \mathbb{R}^{m_i \times n_i}$ for $i \in \{1, ..., N\}$, with $n_i, m_i \in \mathbb{N}_0$ for $i \in \{0, 1, ..., N\}$ and $m_{N+1} \in \mathbb{N}_0$.

This LP structure is quite general and can also be found in many applications beyond energy system modeling. Moreover, arrowhead LPs include important problem classes such as primal block-angular (no linking constraints), dual block-angular (no linking variables), or staircase matrices [**?**]. Primal block-angular matrices arise for instance from two-stage stochastic optimization problems [19]. In particular for two-stage stochastic optimization problems there has been much work on specialized (parallel) algorithms, see e.g. [3, 9, 18, 20], and powerful interior-point solvers for distributed-memory systems such as OOPS [16] and PIPS-IPM [22] exist. This article will present an extension of this work to arrowhead LPs.

A common feature of linear energy system models is the large number of linking constraints and variables (often more than 100 000). This linkage renders straightforward extensions of previous work on stochastic optimization problems prohibitive. However, one often observes that the majority of linking constraints and variables of energy system models only link two (or at least only few) consecutive diagonal blocks. A major contribution of this article is the development and implementation of algorithms that exploit this property and can efficiently handle problems with more than 300 000 linking constraints and variables. The new algorithms are embedded within an interior-point method and the core is constituted by a preconditioned Schur complement decomposition. The implementation of this framework is based on PIPS-IPM.

The computational evaluation of this article concentrates on one specific energy system model: ELMOD. This electricity market model assumes a nodal pricing market regime and finds the cost optimal economic power plant dispatch and transmission grid flows to serve the market's nodal demand. In contrast to a zonal pricing market models that focus only on commercial flows, nodal pricing already includes physical transmission constraints at the stage of market clearing to avoid redispatch activities arising from congestions in the grid topology. Model-based representations of nodal pricing markets likely become large-scale due to the inclusion of the entire high voltage grid, detailed technical modeling of generation units, and a highly-resolved time dimension to represent local diurnal and seasonal variations of renewable feed-in and electricity demand. Consequently, the determination of the market equilibrium is a highly complex task.

## 1.1 Preliminaries

Linking constraints, i.e. the constraints (5), that have only 0 entries in all $U$ matrices apart from possibly $U_0, U_i, U_{i+1}$ for exactly one $i \in \{1, ..., N\}$ will be called *local linking constraints*. The remaining linking constraints will be called *global linking constraints*. Linking variables are classified equivalently (with respect to the $T$ matrices).

While this article focuses on providing a mathematical description of new parallel algorithms, the latter are based on the availability of distributed memory together with certain messaging protocols. The implementation for this article relies on the de facto communication standard in distributed-memory programming: *MPI* [7] (Message-Passing Interface). MPI provides implemen-

tations for most common actions and communications in parallel programming, mostly for point-to-point and collective communication between the different (parallel) processes. These processes are referred to as *MPI processes*. Each MPI process is assigned a unique identifier in $\{0, ..., N-1\}$, referred to as *rank*, where $N$ is the total number of MPI processes used—or rather the number of MPI processes in a certain process group, but this distinction is not needed in this article. In the following, the most important MPI operations are point-to-point communication and the collective operation Reduce. Point-to-point communication involves just two specific MPI processes. A typical operation is *Send*, where one MPI process sends data to another one. Collective communication, on the other hand, involves all MPI processes (again, in general within a certain process group). The collective communication *Reduce* takes data from all MPI processes, performs an operation (such as summing up), and stores the result on one specified MPI process. In this article, Reduce is commonly used to sum up vectors and matrices stored on different MPI processes. A variation of Reduce is *Allreduce*, where the result of the collective operation (e.g. the sum of vectors) is stored on all MPI processes.

# 2 Modeling energy system problems as linear programs

Linear programming is extensively used in a wide range of both research and industry applications. This section shows that linear programming is also a popular tool for modeling energy system problems. It will also be demonstrated how an arrowhead constraint matrix results from common energy system models.

## 2.1 ELMOD: Qualitative description

ELMOD can be classified as a linear optimization model representing the European electricity markets by the use of a nodal pricing market clearing. It serves to investigate the interactions between power generation and grid infrastructure in the electricity sector both, in terms of operation as well as investment needs. For this purpose, the European transmission grid with its underlying generation fleet and demand structure is modeled and assigned to each transmission grid node. In order to keep the model in a linear fashion and determine load flows analytically, a DC load flow approach is utilized. In its basic version the model is applied to analyze numerous questions about the impact of growing shares of renewable energies on the operation of the European transmission grid and the dispatch of conventional power plants. Subject of further research that has been conducted relates to the design of electricity markets, such as optimal market zone configuration, congestion management and security of supply with a geographical focus on Europe.

The bottom-up model determines the optimal economic dispatch and power flow to serve the nodal electricity demand. The necessary generation and transport of electricity is subject to several dispatch and transmission constraints that are related to the technological representation of power plants and physical characteristics of load flow. The underlying power plant fleet comprises conventional generation, combined heat and power units, renewable energy based generators as well as storage applications such as pumped storage plants and

reservoir storages. The entire model's generation and transmission infrastructure is geographically referenced allowing for further applications with a strong regional focus. Due to the high granularity of the model, as of now, it has been applied with a reduced time dimension based on time aggregation methods, such as utilizing typical days or time slices, or other simplifications that come with a limited foresight, such as a rolling horizon approach. However, the underlying database contains fundamental input data for 8760 hours of different reference years.

## 2.2 ELMOD: Mathematical formulation

In the following, ELMOD is described mathematically.

ELMOD has the following indices and sets.

- Time steps $t \in T := \{1, 2, ..., 8760\}$
- Countries $c \in C$
- HVDC line $d \in D$
- Renewable technologies $i \in I := \{$"Biomass", "Geothermal", "Landfill gas", "PV", "Sewage gas", "Wind offshore", "Wind onshore"$\}$
- AC line $l \in L$
- Grid nodes $n \in N$
  Slack node/nodes $k \in K \subset N$
- Power plants $p \in P$
  Renewable energy capacities $r \in R \subset P$
  Pumped storage power plants (PSPs) $s \in S \subset P$
  Reservoir power plants $y \in Y \subset P$

Note that while sets $T$ and $I$ remain the same for all ELMOD instances in this analysis, set $C$, $D$, $L$, $N$ and $P$ depend on the geographical scope of each instance.

ELMOD has the following parameters:

- Power plant characteristics:
  Maximum power output (installed capacity in MW) $g_p^{\max} \in \mathbb{R}_{>0}$
  Availability of power plant $a_{t,p} \in [0, 1]$
  Ramp rate up $r_p^{\text{up}} \in [0, 1]$
  Ramp rate down $r_p^{\text{down}} \in [0, 1]$
  Efficiency of power plant $\eta_s \in [0, 1]$
  Maximum charging capacity of PSP (MW) $g_s^{\text{charge}} \in \mathbb{R}_{>0}$
  Storage capacity (MWh) $stor_s \in \mathbb{R}_{>0}$
  Full load hours of reservoirs (hours) $f_y \in \mathbb{R}_{>0}$
- Costs (all in €/MWh):
  Variable (generation) costs $c_{t,p}^{\text{var}} \in \mathbb{R}_{\geq 0}$
  Lost load $c^{\text{VoLL}} \in \mathbb{R}_{\geq 0}$
  Dumped generation $c^{\text{DG}} \in \mathbb{R}_{\geq 0}$
  Curtailed renewable energy $c^{\text{curt}} \in \mathbb{R}_{\geq 0}$

- Power line characteristics:

  Element of the line susceptance matrix $\mathbf{H} \in \mathbb{R}^{|L|\times|N|}$

  Element of the nodal susceptance[1] matrix $\mathbf{B} \in \mathbb{R}^{|N|\times|N|}$

  Capacity limit of AC line (MW) $cap_l^{\mathrm{AC}} \in \mathbb{R}_{>0}$

  Capacity limit of HVDC line (MW) $cap_l^{\mathrm{HVDC}} \in \mathbb{R}_{>0}$

- Exogenous time series (MW):

  Available aggregated generation for renewable technologies in a country $g_{t,c,i} \in \mathbb{R}_{\geq 0}$

  Demand per node $d_{t,n} \in \mathbb{R}_{\geq 0}$

ELMOD has the following variables:

- Total costs $TC \in \mathbb{R}_{\geq 0}$
- Generation costs $CG_{t,c} \in \mathbb{R}_{\geq 0}$
- Infeasibility costs $CI_{t,c} \in \mathbb{R}_{\geq 0}$
- Curtailment costs $CC_{t,c} \in \mathbb{R}_{\geq 0}$
- Power plant generation $G_{t,p} \in \mathbb{R}_{\geq 0}$
- Power demanded by PSP $PUMP_{t,s} \in \mathbb{R}$
- Storage level $SL_{t,n} \in \mathbb{R}_{\geq 0}$
- Summation of reservoir power generation $G_y^{\mathrm{sum}} \in \mathbb{R}_{\geq 0}$
- Dumped demand $DUMP_{t,n}^{\mathrm{dem}} \in \mathbb{R}_{\geq 0}$
- Dumped generation $DUMP_{t,n}^{\mathrm{gen}} \in \mathbb{R}_{\geq 0}$
- Net injection $INJ_{t,n} \in \mathbb{R}$
- Power flow on HVDC $F_{t,d}^{\mathrm{HVDC}} \in \mathbb{R}$
- Power flow on AC line $F_{t,l}^{\mathrm{AC}} \in \mathbb{R}$
- Voltage angle $DELTA_{t,n} \in \mathbb{R}$

The variables and constraints of ELMOD are described on the basis of the following equation system 7. Equation 7a describes the objective function of the model which is the minimization of total costs $TC$. The components of the total costs are shown in Equation 7b - 7d. Each component is created for every country and time step separately. The objective function sums up the components for all countries and time steps.

The generation costs $CG_{t,c}$ are the summation of the generation of all power plants $G_{t,p}$ within the respective country ($mp(c)$), multiplied with the time- and plant-dependent variable costs (7b). The infeasibility costs $CI_{t,c}$ are computed by multiplying the amounts of dumped demand or generation, $DUMP_{t,n}^{\mathrm{dem}}$ and

[1] In this analysis the susceptance values in matrix $\mathbf{H}$ and $\mathbf{B}$ already contain the squared voltage levels of the respective lines. Thus power flows are calculated as a product of susceptance and voltage angle differences (see Equations 7i and 7j).

$DUMP^{\text{gen}}_{t,n}$, with the corresponding cost factors (7c).[2] Curtailment costs $CC_{t,c}$ only occur when the available aggregated generation for a renewable technology in a country surpasses the sum of actual generation $G_{t,r}$ for all renewable power plants of a specific technology in that country ($mp(c,i)$) (7d).

Equation 7e describes the power balance for each node and time step. The aggregated demand in addition to the net injection $INJ_{t,n}$ have to be satisfied. The variables that contribute to meeting the left-hand side demand are node-specific dumped demand and the generation of power plants at the respective node ($mp(n)$). Dumped generation as well as power demanded by PSPs $PUMP_{t,s}$ connected to the node ($mp(n)$) contribute negatively to satisfying the demand. How the positive or negative flows of HVDC lines $F^{\text{HVDC}}_{t,d}$ contribute to the nodal balance depends on the start and end points of the line. All HVDC lines that start at the node ($mds(n)$) contribute to the nodal balance with reversed sign of their flows. The opposite is true for all HVDC lines that end at the node ($mde(n)$).

The generation of power plant is limited by the product of its maximum capacity and a time- and plant-dependent availability factor (Equation 7f). For PSPs Equation 7g is additionally imposed to limit the amount of pumped energy to the charging capabilities of the plant. Also, the storage level of the PSP cannot surpass its maximum storage capacity (Equation 7h).

The flows on AC power lines $F^{\text{AC}}_{t,l}$ and the net injections at each node are computed according to Equation 7i and 7j. This represents a DC load flow approach (cf. [26]). The voltage angles of all nodes $DELTA_{t,n}$ are endogenous variables. The exception is the voltage angle of the slack node $k$ that is automatically set to zero (Equation 7k). The line flows of AC and HVDC lines are limited according to their maximum capacity, both for positive and negative flows (Equation 7l - 7o).

With the exception of the objective function, all equations described so far can be distinctly ascribed to a diagonal block as they have no summations across time steps or inter-temporal links.

The model has three different inter-temporal relationships (Equation7p - 7r). The storage level of the current period takes into account the storage level of the previous period $SL_{t-1,s}$ as well as the generated and pumped energy (Equation 7p). Equation 7s ensures that the storage level in the first and last period are equal to half of the storage capacity. For conventional power plants, a restriction for ramping up and down can apply. Equation 7q and 7r stipulate that the change in generation between two time steps cannot exceed the allowable change in output, the product of ramp rate and capacity. All three equations are local linking constraints.

Lastly, the aggregated generation of a reservoir power plant $G^{\text{sum}}_{y}$ is the summation of its generation levels over all time steps (Equation 7t), which makes it a global linking constraint. Total costs $TC$ and $G^{\text{sum}}_{y}$ are the only global variables. In Equation 7u each variable $G^{\text{sum}}_{y}$ is limited by the product of full load hours, capacity and the fraction of considered time steps in relation to the entire year. Because the equation limits a global variable it is assigned

[2]These variables are usually equal to zero. Values deviating from zero serve to identify infeasibilities.

to the first stage.

$$\min \quad TC = \sum_{t,c}(CG_{t,c} + CI_{t,c} + CC_{t,c}) \tag{7a}$$

s.t.

$$CG_{t,c} = \sum_{p \in mp(c)} G_{t,p} \cdot c^{\text{var}}_{t,p} \qquad \forall t \in T \ \forall c \in C \tag{7b}$$

$$CI_{t,c} = \sum_{n \in mn(c)} (DUMP^{\text{dem}}_{t,n} \cdot c^{\text{VoLL}} + DUMP^{\text{gen}}_{t,n} \cdot c^{\text{DG}}) \qquad \forall t \in T \ \forall c \in C \tag{7c}$$

$$CC_{t,c} = \sum_{i} \Big[ (g_{t,c,i} - \sum_{r \in mp(c,i)} G_{t,r}) \cdot c^{\text{curt}} \Big] \qquad \forall t \in T \ \forall c \in C \tag{7d}$$

$$d_{t,n} + INJ_{t,n} = \sum_{d \in mde(n)} F^{\text{HVDC}}_{t,d} + DUMP^{\text{dem}}_{t,n} - \sum_{d \in mds(n)} F^{\text{HVDC}}_{t,d} - DUMP^{\text{gen}}_{t,n} - \sum_{s \in mp(n)} PUMP_{t,s} + \sum_{p \in mp(n)} G_{t,p} \qquad \forall t \in T \ \forall n \in N \tag{7e}$$

$$G_{t,p} \leq g^{\max}_{p} \cdot a_{t,p} \qquad \forall t \in T \ \forall p \in P \tag{7f}$$

$$PUMP_{t,s} \leq g^{\text{charge}}_{s} \qquad \forall t \in T \ \forall s \in S \tag{7g}$$

$$SL_{t,s} \leq stor_{s} \qquad \forall t \in T \ \forall s \in S \tag{7h}$$

$$F^{\text{AC}}_{t,l} = \sum_{n \in N} (h_{l,n} \cdot DELTA_{t,n}) \qquad \forall t \in T \ \forall l \in L \tag{7i}$$

$$INJ_{t,n} = \sum_{q \in N} (b_{n,q} \cdot DELTA_{t,q}) \qquad \forall t \in T \ \forall n \in N \tag{7j}$$

$$DELTA_{t,k} = 0 \qquad \forall t \in T \ \forall k \in K \tag{7k}$$

$$F^{\text{AC}}_{t,l} \leq cap^{\text{AC}}_{l} \qquad \forall t \in T \ \forall l \in L \tag{7l}$$

$$-F^{\text{AC}}_{t,l} \leq cap^{\text{AC}}_{l} \qquad \forall t \in T \ \forall l \in L \tag{7m}$$

$$F^{\text{HVDC}}_{t,d} \leq cap^{\text{HVDC}}_{d} \qquad \forall t \in T \ \forall d \in D \tag{7n}$$

$$-F_{t,d}^{\mathrm{HVDC}} \leq cap_d^{\mathrm{HVDC}} \qquad \forall t \in T \ \ \forall d \in D \tag{7o}$$

$$SL_{t,s} = SL_{t-1,s} + PUMP_{t,s} \cdot \eta_s - G_{t,s} \qquad \forall t \in T \setminus \{1\}, \ \forall s \in S \tag{7p}$$

$$G_{t,p} - G_{t-1,p} \leq r_p^{\mathrm{up}} \cdot g_p^{\max} \qquad \forall t \in T \setminus \{1\}, \ \forall p \in P \tag{7q}$$

$$-G_{t,p} + G_{t-1,p} \leq r_p^{\mathrm{down}} \cdot g_p^{\max} \qquad \forall t \in T \setminus \{1\}, \ \forall p \in P \tag{7r}$$

$$SL_{t,s} = 0.5 \cdot stor_s \qquad \forall t \in \{1, 8760\} \ \ \forall s \in S \tag{7s}$$

$$G_y^{\mathrm{sum}} = \sum_t G_{t,y} \qquad \forall y \in Y \tag{7t}$$

$$G_y^{\mathrm{sum}} \leq f_y \cdot g_y^{\max} \qquad \forall y \in Y \tag{7u}$$

## 2.3 Arrowhead structure in other energy system models

The arrowhead structure of the constraint matrix is not a peculiarity of the ELMOD model, but can also be observed in several other linear energy system models. Indeed, it arises naturally if the model considers a discrete time horizon. Moreover, one typically observes a large number of local linking constraints (or variables), which for instance originate from modeling short term storages between time steps such as pumped storages or batteries and further flexibility options related to the demand side such as load shifting, but also from the representation of technical power plant dispatch limitations such as ramping constraints. Prominent examples of such linear energy system models are the REMix [13] or the Balmorel [31] model.

State-of-the-art energy system models are usually highly complex—having been developed over years or even decades by several people. This complexity motivates the need for a simplified energy system model that maintains relevant parts of the model structure that can be found in many energy system models, but is at the same time compact and comprehensive. Within, the BEAM-ME project[3] modeling experts from GAMS[4] together with energy system modelers from the German Aerospace Center[5] developed such a simplified, but representative, energy system model, called *SIMPLE*. While SIMPLE lacks many details that are considered in full-fledged energy system models, it is easily adaptable in size (e.g. number of variables or number of diagonal blocks) and thus highly useful for testing new solution approaches. Also, it contains far less redundancies than many large-scale energy system instances (caused by complex and highly intricate models), which allows for a more revealing comparison of new solution methods with state-of-the-art general LP solvers—which include powerful presolving routines that can already drastically reduce the run time.

[3] http://www.beam-me-projekt.de/beam-me/EN/Home/home_node.html
[4] https://www.gams.com/
[5] https://www.dlr.de

A core concept of SIMPLE is the automatic generation of input data. The SIMPLE models come with a data generator that can be parameterized to generate data instances of different size. All the data is computed by randomizing standard basic time series that provide the corresponding data for a standard region. In addition, the regions are placed on a 1000x1000 km grid and a network of transmission links is computed that connects different regions under consideration of distances between the regions. Note that even though there is randomization involved in the automated data generation, the process is deterministic. The data generator always produces data instances that cover an entire year in hourly resolution. In addition, the SIMPLE models allow the user to work with further customized *model data*. This includes the possibility to change the time resolution, which means that hourly time steps from the input data can be aggregated or disaggregated for the actual model data. Furthermore, the time horizon to be considered can be controlled via parameters. Importantly, any linear program created by SIMPLE shows an arrowhead structure. Note that none of the authors of this article have been involved in the development of SIMPLE and it should thus not be considered as a contribution of this article. However, SIMPLE will be used to on the one hand demonstrate the scalability of the newly developed solver and, on the other hand, the applicability of the new solver beyond the ELMOD model.

# 3 Exploiting the structure: A specialized parallel interior-point algorithm

For general LPs the two major algorithmic classes are simplex and interior-point methods, see e.g. [29]. Interior-point methods are often more successful for large problems, and they offer more potential for parallelization, since the main computational effort usually goes into factorizing matrices. In this article we use infeasible primal-dual interior-point methods [32]. The following describes the linear systems to be solved within a primal-dual interior-point method. For a derivation of these systems (and a detailed description of interior-point methods) the reader is referred to [32]. Consider the following LP in *standard form*:

$$\begin{aligned} \min \quad & c^T x && (8)\\ \text{s.t.} \quad & Ax = b && (9)\\ & x \geq 0 && (10) \end{aligned}$$

with $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$, and its *dual*:

$$\begin{aligned} \max \quad & b^T y && (11)\\ \text{s.t.} \quad & A^T y + s = c && (12)\\ & s \geq 0. && (13) \end{aligned}$$

If both exist, the optimal objective values of an LP (referred to as the *primal*) and its dual LP coincide [29]. Primal-dual interior-point methods provide optimal solutions to both the primal and dual problem. In each iteration of a

primal-dual interior-point algorithm it is sufficient to solve the so-called *augmented system*:

$$\begin{bmatrix} -X^{-1}S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \end{bmatrix}, \tag{14}$$

where $X$ and $S$ denote diagonal matrices whose diagonals are equal to the current iterates $x$ and $s$, respectively. Note that the matrix is in general symmetric indefinite. It is possible to bring any LP into standard form, but in an actual implementation it is (for reasons of efficiency [2, 32]) preferable to keep an LP in general form, including inequalities and variable bounds. For ease of presentation, however, in this article it will be assumed that an LP is given in standard form.

## 3.1 Decomposing the problem: a parallel Schur complement approach

For the arrowhead LP from Section 1 the augmented system (14) takes the form

$$\begin{bmatrix} D_0 & & & & T_0^T & T_1^T & \cdots & T_N^T & U_0^T \\ & D_1 & & & & W_1^T & & & U_1^T \\ & & \ddots & & & & \ddots & & \vdots \\ & & & D_N & & & & W_N^T & U_N^T \\ T_0 & & & & & & & & \\ T_1 & W_1 & & & & & & & \\ \vdots & & \ddots & & & & & & \\ T_N & & & W_N & & & & & \\ U_0 & U_1 & \cdots & U_N & & & & & \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta x_1 \\ \vdots \\ \Delta x_N \\ \Delta y_0 \\ \Delta y_1 \\ \vdots \\ \Delta y_N \\ \Delta y_{N+1} \end{bmatrix} = \begin{bmatrix} r_{x_0} \\ r_{x_1} \\ \vdots \\ r_{x_N} \\ r_{y_0} \\ r_{y_1} \\ \vdots \\ r_{y_N} \\ r_{y_{N+1}} \end{bmatrix} \tag{15}$$

where $D_i := -X_i^{-1}S_i$ for $i \in \{0, ..., N\}$. In the following, it will be shown how to solve (15) in a distributed parallel fashion; by using a straightforward extension of an approach that has been used for two-stage stochastic LPs (which have a block-diagonal constraint matrix with linking variables, but without linking constraints). For the latter approach see e.g. [22].

First, by applying a symmetric permutation one can write the augmented system (15) as:

$$\begin{bmatrix} D_1 & W_1^T & & & & 0 & 0 & U_1^T \\ W_1 & 0 & & & & T_1 & 0 & 0 \\ & & \ddots & & & \vdots & \vdots & \vdots \\ & & & D_N & W_N^T & 0 & 0 & U_N^T \\ & & & W_N & 0 & T_N & 0 & 0 \\ 0 & T_1^T & \cdots & 0 & T_N^T & D_0 & T_0^T & U_0^T \\ 0 & 0 & \cdots & 0 & 0 & T_0 & 0 & 0 \\ U_1 & 0 & \cdots & U_N & 0 & U_0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_N \\ \Delta y_N \\ \Delta x_0 \\ \Delta y_0 \\ \Delta y_{N+1} \end{bmatrix} = \begin{bmatrix} r_{x_1} \\ r_{y_1} \\ \vdots \\ r_{x_N} \\ r_{y_N} \\ r_{x_0} \\ r_{y_0} \\ r_{y_{N+1}} \end{bmatrix} \tag{16}$$

This system again shows an arrowhead form, namely

$$\begin{bmatrix} K_1 & & & B_1 \\ & \ddots & & \vdots \\ & & K_N & B_N \\ B_1^T & \cdots & B_N^T & K_0 \end{bmatrix} \begin{bmatrix} \Delta z_1 \\ \vdots \\ \Delta z_N \\ \Delta z_0 \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \\ b_0 \end{bmatrix} \tag{17}$$

where

$$K_i = \begin{bmatrix} D_i & W_i^T \\ W_i & 0 \end{bmatrix},\ K_0 = \begin{bmatrix} D_0 & T_0^T & U_0^T \\ T_0 & 0 & 0 \\ U_0 & 0 & 0 \end{bmatrix},\ B_i = \begin{bmatrix} 0 & 0 & U_i^T \\ T_i & 0 & 0 \end{bmatrix} \tag{18}$$

for $i \in \{1, ..., N\}$ and the $\Delta z_i$ and $b_i$ are set accordingly. This system can be solved by the Schur complement approach [33] if one consider the entire system as a (symmetric indefinite) saddle-point system with the $(1, 1)$ block constituted by the $K_i$ matrices, the $(1, 2)$ block by the $B_i$ matrices, and the $(2, 2)$ block by $K_0$. A perhaps more intuitive way is to think of a block-Gaussian elimination. Assuming that all $K_i$ have full rank, one obtains the following procedure:

1. Multiply each row $i = 1, ..., N$ of (17) by $-B_i^T K_i^{-1}$.
2. Sum up all rows.
3. Solve $\left(K_0 - \sum_{i=1}^N B_i^T K_i^{-1} B_i\right)\Delta z_0 = b_0 - \sum_{i=1}^N B_i^T K_i^{-1} b_i$.
4. For each row $i = 1, ..., N$ insert $\Delta z_0$ and compute $\Delta z_i$.

Note that in this article (and in fact in arguably all computational sciences) the inverse of a matrix is just used for ease of notation. For reasons of efficiency, in the actual implementation a factorization of the matrix is used. Since the $K_i$ are symmetric, but indefinite, an $LDL^T$ factorization is used—with $L$ lower triangular, and $D$ block-diagonal with blocks of dimension $1 \times 1$ or $2 \times 2$, see also [32].

From a more computational (parallel) view the Schur complement approach can also we written in the following form:

$$V_i = B_i^T K_i^{-1} B_i \qquad i = 1, ..., N \tag{19}$$

$$C = K_0 - \sum_{i=1}^N V_i \tag{20}$$

$$w_i = B_i^T K_i^{-1} b_i \qquad i = 1, ..., N \tag{21}$$

$$\Delta z_0 = C^{-1}(b_0 \sum_{i=1}^N w_i) \tag{22}$$

$$v_i = B_i \Delta z_0 \qquad i = 1, ..., N \tag{23}$$

$$\Delta z_i = K^{-1}(b_i - v_i) \qquad i = 1, ..., N \tag{24}$$

An important feature of the Schur complement decomposition is the possibility to distribute the LP (by blocks) among the MPI processes, with no process needing to store the entire problem. For ease of presentation we will in the following assume that $N$ MPI processes are used. In this case each process

$i \in \{0, ..., N-1\}$ only needs to store and access the matrices and vectors with index $i+1$, and those with index 0. In this way one can tackle problems that are too large to even be stored in the main memory of a single desktop machine.

## 3.2 Computing the global Schur complement

The *Schur complement* of system (17) is

$$C = K_0 - \sum_{i=1}^{N} B_i^T K_i^{-1} B_i, \tag{25}$$

which is symmetric indefinite. Note that one can also consider each summand in (25) as a Schur complement of a suitable (augmented) symmetric matrix [23]. Thus, we will also refer to (25) as the *global Schur complement*, and to each $B_i^T K_i^{-1} B_i$ as a *local Schur complement*. If not noted otherwise, the term Schur complement will refer to (25), however.

Considering the Schur complement decomposition algorithm, one observes that the only tasks that are not performed independently are the formation (22) of and the solve operation (23) with the Schur complement. Moreover, one observes that the size of the Schur complement depends on the number of linking constraints and linking variables. For some energy system models considered in this article there are more than 300 000 linking constraints and variables, which makes it intractable to compute and store the Schur complement as a dense matrix, which is the common procedure for two-stage stochastic optimization approaches. As already noticed, however, in many linear energy system models the majority of linking constraints and variables are local. Indeed, as the number of blocks grows (for instance by considering a longer time frame), the number of local linking variables and constraints usually grow linearly, while the number of global linking constraints and variables stays constant. For simplicity, only the case of local linking constraints will be demonstrated, and it will be assumed that there are few global linking constraints and few linking variables. Consider again the block structure from Section 1, but split the linking constants in global and local ones, namely:

$$\begin{aligned}
\min \quad & \sum_{i=0}^{N} c^T x_i \\
\text{s.t.} \quad & T_0 x_0 & & & & & & & = h_0 && (26)\\
& T_1 x_0 & + & W_1 x_1 & & & & & = h_1 && (27)\\
& T_2 x_0 & + & & W_2 x_2 & & & & = h_2 && (28)\\
& \vdots & & & & \ddots & & & \vdots & &\\
& T_N x_0 & + & & & & & W_N x_N & = h_N && (29)\\
& F_0 x_0 & + & F_1 x_1 & + & F_2 x_2 & \cdots & F_N x_N & = h_{N+1} && (30)\\
& G_0 x_0 & + & G_1 x_1 & + & G_2 x_2 & \cdots & G_N x_N & = h_{N+2} && (31)\\
& x_0, & & x_1, & & x_2, & \cdots & x_N & \geq 0 && (32)
\end{aligned}$$

such that $G_i \in \mathbb{R}^{m_G \times n_i}$ with arbitrary structure and $F_i \in \mathbb{R}^{m_F \times n_i}$ for all $i \in \{0, 1, ..., N\}$; with $m_F, m_G \in \mathbb{N}$. Furthermore, for $i \geq 1$ the $F_i$ are assumed

to be of the form

$$F_1 = \begin{bmatrix} Z_1 \\ 0 \\ \vdots \\ \end{bmatrix}, \; F_2 = \begin{bmatrix} Z_1' \\ Z_2 \\ 0 \\ \vdots \\ \end{bmatrix}, \; F_3 = \begin{bmatrix} 0 \\ Z_2' \\ Z_3 \\ 0 \\ \vdots \\ \end{bmatrix}, ..., \; F_N = \begin{bmatrix} 0 \\ \vdots \\ \\ \\ Z_{N-1}' \end{bmatrix},$$

with $Z_i \in \mathbb{R}^{l_i \times n_i}$, $Z_i' \in \mathbb{R}^{l_i \times n_{i+1}}$, $l_i \in \mathbb{N}_0$ for $i \in \{1, ..., N-1\}$. By defining

$$U_i := \begin{bmatrix} F_i \\ G_i \end{bmatrix}$$

for all $i \in \{1, ..., N\}$, one can apply the same Schur decomposition approach as in Section 3.1. In the following it will be shown how the non-zero pattern of the $F_i$ matrices impacts the structure of the Schur complement (25). To this end, consider for a $i \in \{1, ..., N\}$ the matrix $B_i^T K_i^{-1} B_i$. We write

$$K_i^{-1} = \begin{bmatrix} \tilde{K}_{1,1}^i & \tilde{K}_{1,2}^i \\ \tilde{K}_{2,1}^i & \tilde{K}_{2,2}^i \end{bmatrix}$$

such that the dimensions of the submatrices are conform with the following operations. With this notation one obtains:

$$B_i^T K_i^{-1} B_i = \begin{bmatrix} 0 & T_i^T \\ 0 & 0 \\ F_i & 0 \\ G_i & 0 \end{bmatrix} \begin{bmatrix} \tilde{K}_{1,1}^i & \tilde{K}_{1,2}^i \\ \tilde{K}_{2,1}^i & \tilde{K}_{2,2}^i \end{bmatrix} \begin{bmatrix} 0 & 0 & F_i^T & G_i^T \\ T_i & 0 & 0 & 0 \end{bmatrix} \tag{33}$$

$$= \begin{bmatrix} 0 & T_i^T \\ 0 & 0 \\ F_i & 0 \\ G_i & 0 \end{bmatrix} \begin{bmatrix} \tilde{K}_{1,2}^i T_i & 0 & \tilde{K}_{1,1}^i F_i^T & \tilde{K}_{1,1}^i G_i^T \\ \tilde{K}_{2,2}^i T_i & 0 & \tilde{K}_{2,1}^i F_i^T & \tilde{K}_{2,1}^i G_i^T \end{bmatrix} \tag{34}$$

$$= \begin{bmatrix} T_i^T \tilde{K}_{2,2}^i T_i & 0 & T_i^T \tilde{K}_{2,1}^i F_i^T & T_i^T \tilde{K}_{2,1}^i G_i^T \\ 0 & 0 & 0 & 0 \\ F_i \tilde{K}_{1,2}^i T_i & 0 & F_i \tilde{K}_{1,1}^i F_i^T & F_i \tilde{K}_{1,1}^i G_i^T \\ G_i \tilde{K}_{1,2}^i T_i & 0 & G_i \tilde{K}_{1,1}^i F_i^T & G_i \tilde{K}_{1,1}^i G_i^T \end{bmatrix} \tag{35}$$

If (as we assume) most of the linking constraints are local and there are few linking variables, the number of rows of $F_i$ is close to the size of the local Schur complement (35). Thus, the largest part of (35) is $F_i \tilde{K}_{1,1}^i F_i^T$. For $i > 1$ the structure of this matrix is as follows:

$$F_i \tilde{K}_{1,1}^i F_i^T = \begin{bmatrix} \vdots \\ 0 \\ Z_{i-1}' \\ Z_i \\ 0 \\ \vdots \end{bmatrix} \tilde{K}_{1,1}^i \begin{bmatrix} \cdots & 0 & Z_{i-1}'^T & Z_i^T & 0 & \cdots \end{bmatrix}$$

$$= \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \iddots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & Z'_{i-1}\tilde{K}^i_{1,1}Z'^T_{i-1} & Z'_{i-1}\tilde{K}^i_{1,1}Z_i^T & 0 & \cdots \\ \cdots & 0 & Z_i\tilde{K}^i_{1,1}Z'^T_{i-1} & Z_i\tilde{K}^i_{1,1}Z_i^T & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \iddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Consequently, after suitable symmetric permutation, the global Schur complement (17) without $K_0$ is of symmetric, doubly bordered, block-tridiagonal form. Thus there exists a permutation matrix $P$ such that:

$$P^T\big(\sum_{i=1}^{N} B_i^T K_i^{-1} B_i\big)P = \begin{bmatrix} M_1 & H_1^T & & & & E_1^T & 0 \\ H_1 & M_2 & H_2^T & & & E_2^T & 0 \\ & H_2 & \ddots & \ddots & & \vdots & \vdots \\ & & \ddots & & H_{N-2}^T & E_{N-2}^T & 0 \\ & & & H_{N-2} & M_{N-1} & E_{N-1}^T & 0 \\ E_1 & E_2 & \cdots & E_{N-1} & E_{N-1} & M_0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 \end{bmatrix} \tag{36}$$

where the submatrices are defined as follows. First, for $i \in \{1, ..., N-1\}$:

$$M_i = Z_i\tilde{K}^i_{1,1}Z_i^T + Z'_i\tilde{K}^{i+1}_{1,1}Z'^T_i \tag{37}$$

$$E_i = \begin{bmatrix} G_i\tilde{K}^i_{1,1}Z_i^T + G_{i+1}\tilde{K}^{i+1}_{1,1}Z'^T_i \\ T_i^T\tilde{K}^i_{2,1}Z_i^T + T_{i+1}^T\tilde{K}^{i+1}_{2,1}Z'^T_i \end{bmatrix}. \tag{38}$$

Second, for $i \in \{1, ..., N-2\}$:

$$H_i = Z_{i+1}\tilde{K}^{i+1}_{1,1}Z'^T_i. \tag{39}$$

And finally

$$M_0 = \sum_{i=1}^{N} \begin{bmatrix} G_i\tilde{K}^i_{1,1}G_i^T & G_i\tilde{K}^i_{1,2}T_i \\ T_i^T\tilde{K}^i_{2,1}G_i^T & T_i^T\tilde{K}^i_{2,2}T_i \end{bmatrix}. \tag{40}$$

In a similar way, one can precompute the non-zero pattern for the case of linking variables that link only two blocks—in this case there would be another block-wise structure within the parts of (36) occupied by the $E_i$ and $M_0$ blocks. Note that already for the case of both local equality and local inequality linking constraints (which indeed occur for the energy system models considered in this article), one obtains a different, more complicated structure of the Schur complement. But still, this structure can be precomputed.

Unless there is an additional special structure within the $T_i$ or $G_i$ matrices, the submatrices $E_i, H_i$ and $M_i$ in (36) are dense. Still, the matrix (36) can be very sparse, as quantified by the following observation. While the actual Schur complement also includes $K_0$ (17), this matrix is assumed to be sparse and is therefore neglected in the following. Recall that $n_0$ denotes the number of linking variables, and $m_G$ the number of global linking constraints.

**Observation 1.** *The number of non-zeroes in* (36) *is at most*

$$\sum_{i=1}^{N-1} \ell_i^2 + 2\sum_{i=1}^{N-2} \ell_i \ell_{i+1} + 2\sum_{i=1}^{N-1} \ell_i (m_G + n_0) + (m_G + n_0)^2. \tag{41}$$

*In particular, if* $\ell_1 = \ell_2 = \ldots = \ell_{N-1} =: \ell$*, then the number of non-zeroes in* (36) *can be expressed as the affine map*

$$f(N) := 3(N - \tfrac{5}{3})\ell^2 + 2(N-1)\ell(m_G + n_0) + (m_G + n_0)^2. \tag{42}$$

*Proof.* Follows from (37), (38), (39), and (40). □

Note that in an actual implementation one only needs to compute and store the upper or lower triangular part of the Schur complement, due to its being symmetric. As already pointed out, typically the number of global linking constraints and variables does not increase if more blocks are considered. Only the number of local linking constraints increases if one increases the number of blocks—and this increase is linear. In this case, Observation 1 implies that the number of non-zeroes of the Schur complement grows only linearly—compared to the quadratic non-zeroes growth of a general (dense) matrix with its size. While such a linear growth is an encouraging behavior, a large number of global linking constraints ($m_G$) or variables ($n_0$), or a large number of local linking constraints per block ($\ell$) might still render the number of non-zeroes in the Schur complement prohibitively large even for a small number of blocks ($N$).

Knowing that sufficient sparsity exists, one should use a sparse (direct) linear solver for the Schur complement, which can be significantly faster and requires far less memory than a dense direct solver. However, the knowledge of the actual non-zero pattern of the Schur complement is also highly important, as it allows one to drastically reduce both the memory usage and the MPI communication. Moreover, also the fact that all submatrices in (36) apart from $M_0$ are the sum of at most two submatrices (that can be computed independently by individual MPI processes) will be put to use in the next section.

### 3.3 A distributed preconditioner for the Schur complement

If the number of linking constraints and variables is large, or if there are many global linking constraints and variables, the factorization and solution of the global Schur complement can still be prohibitively time-consuming (or even impossible due to memory restrictions). One possible remedy is to not compute the Schur complement explicitly, but to use an iterative approach, e.g. a Krylov subspace method [25]. Iterative methods only require the result of matrix-vector multiplications with the matrix of the linear system, which in the case of the Schur complement (25) can be written as

$$\big(K_0 - \sum_{i=1}^{N} B_i^T K_i^{-1} B_i\big)x = K_0 x - \sum_{i=1}^{N} B_i^T K_i^{-1} B_i x. \tag{43}$$

Therefore, the Schur complement does not need to be formed explicitly; computing the summands $B_i^T K_i^{-1} B_i$ is sufficient. However, the practical success of

iterative methods is usually highly dependent on effective preconditioning of the linear system [30]. Even more so for linear systems occurring in interior-point algorithms, which tend to become highly ill-conditioned during the later stages of the solution phase (due to both very large and close to 0 entries on the diagonal). Efficient preconditioners are often tailored to a specific application or model, but the aim of this article is to develop a solver that is also usable for others linear energy system models (apart from ELMOD).

One observes that the diagonal elements of the global Schur complement(25) are often large (in absolute values). However, they can also be 0, thus a simple diagonal preconditioning is not promising. However, a preconditioner for a similar structure is suggested in [5]. The authors describe a Schur complement based LU factorization of a linear system arising from circuit simulation. Although in this application the Schur complement is dense and small (of size less than 1000), it also shows many large diagonal entries. In [5] the authors suggest a preconditioner based on discarding small elements of the Schur complement $C$. The preconditioner $\tilde{C}$ is defined by

$$\tilde{c}_{ij} := \begin{cases} c_{ij}, & \text{if } |c_{ij}| > t \max\{|c_{ii}|, |c_{jj}|\} \\ 0, & \text{otherwise} \end{cases} \tag{44}$$

with $t \in [0, 1)$; in [5] the value $t := 0.02$ is recommended, but we use more conservative values, see Section 4.1. Note that to the best of the authors knowledge this type of preconditioning has not been described in the literature in the context of interior-point methods yet. Further, it is important to note that since in this article the Schur complement matrix is symmetric, $\tilde{C}$ is symmetric as well.

Considering the structure of the Schur complement matrix, one observes that $\tilde{C}$ can be computed in a distributed fashion, without ever having to form (36) explicitly: First, each MPI process $i \in \{0, ..., N-2\}$ computes $E_i$, $H_i$, and $M_i$ (except for $M_0$) by using point-to-point communication with MPI process $i+1$. Also, the local parts of $T_0, F_0$, and $G_0$ are added. Second, one computes the diagonal by using an Allreduce operation. Now the local parts of $\tilde{C}$ can be computed by each MPI process individually and are gathered to process 0. Finally, a Reduce operation is performed to obtain $M_0$ and the remainder of $\tilde{C}$ is computed on process 0. Computationally, this distributed construction of $\tilde{C}$ is of crucial importance, since it not only reduces the MPI communication, but also circumvents memory bottlenecks.

Yet another modification is to not use the preconditioner $\tilde{C}$ for solving the Schur complement system, but to replace the Schur complement in (20) by $\tilde{C}$ and thereby obtain a preconditioner for the entire augmented system. In this context, one observes that an inexact solution to the Schur complement system (22) has the same residual as the resulting inexact solution to the entire system (17), up to a change of dimension. We formally state this observation in the following. Although it might be considered folklore, we also provide a proof since it will be used subsequently. For ease of presentation we write $z$ instead of $\Delta z$.

**Observation 2.** *Denote by $Mz = b$ the system* (17) *and by $C$ the corresponding*

*Schur complement* (22)*, with* $C \in \mathbb{R}^{k \times k}$*. Let* $\tilde{z}_0 \in \mathbb{R}^k$ *and let*

$$\tilde{r}_C := C\tilde{z}_0 - b_0 + \sum_{i=1}^{N} B_i^T K_i^{-1} b_i \tag{45}$$

*be the resulting residual for the Schur complement system. Denote by* $\tilde{z} = (\tilde{z}_0, \tilde{z}_1, ..., \tilde{z}_N)^T$ *the vector obtained from performing steps* (23) *and* (24) *with* $\tilde{z}_0$ *(instead of* $\Delta z_0$*). It holds that*

$$\tilde{r} := M\tilde{z} - b = (0, 0, ..., 0, \tilde{r}_C)^T. \tag{46}$$

*Proof.* First, one readily verifies that $\tilde{r}$ is 0 in all but (possibly) the last $k$ entries. To this end, let $i \in \{1, ..., N\}$. It follows that

$$K_i\tilde{z}_i + B_i\tilde{z}_0 - b_i = K_iK_i^{-1}(b_i - B_i\tilde{z}_0) + B_i\tilde{z}_0 - b_i = 0. \tag{47}$$

Second, the vector of the last $k$ entries of $\tilde{r}$ is equal to

$$\sum_{i=1}^{N} B_i^T\tilde{z}_i + K_0\tilde{z}_0 - b_0 = \sum_{i=1}^{N} B_i^T K_i^{-1}(b_i - B_i\tilde{z}_0) + K_0\tilde{z}_0 - b_0 \tag{48}$$

$$= C\tilde{z}_0 - b_0 + \sum_{i=1}^{N} B_i^T K_i^{-1} b_i \tag{49}$$

$$\overset{(45)}{=} \tilde{r}_C. \tag{50}$$

Combining this result with (47) yields (46). □

PIPS-IPM uses an incomplete factorization approach for directly computing the local Schur complements [23], which adds some perturbation to the local Schur complements to allow for more efficient pivoting during their computation. As can be seen from Observation 2, such a perturbation does not lead to any non-zero residual entries apart from those corresponding to the global Schur complement (at least in exact computation). However, also the factorization of $K_i$ (which can be implicitly obtained from the incomplete factorization approach) is perturbed. In this case one verifies from (47) that even in exact computation a non-zero residual outside the $\tilde{r}_C$ part occurs. Therefore, iterative refinement needs to be applied for each solve operation with $K_i$—such a solve operation is for example required for the matrix-vector multiplication (43), as $B_i^T K^{-1} B_i$ is only available in perturbed form. Still, one would like to bound the maximum number of iterative refinement steps to obtain a satisfactory load balancing. Therefore, even if one can solve the Schur complement system to sufficiently high precision, there can still be a considerable error in the resulting solution to the entire augmented system. Indeed, the original PIPS-IPM already applies a iterative method for the entire augmented system and uses the Schur complement approach for implicit preconditioning. Consequently, per default we do not use an iterative (preconditioned) approach for solving the Schur complement system, but rather replace the Schur complement matrix $C$ by $\tilde{C}$ in (20) and let the error propagate. In this way, we obtain an implicit preconditioner for the entire augmented system. We describe this procedure in more detail in the following.

In each iteration of the interior-point method, let $K$ be the current augmented matrix. We first compute an implicit factorization of a perturbed version $\tilde{K}$ of $K$ as also shown in Algorithm 1. The perturbation results from the use of $\tilde{C}$ instead of $C$ in (20). The subroutine SPARSIFY in Algorithm 1 applies the operation detailed in (44) to the given submatrices of the Schur complement. Importantly, for this operation only the corresponding diagonal values of the augmented system for these submatrices need to be known. Again, for ease of presentation we ignore $K_0$ in Algorithm 1.

For each solve operation $Kx = b$ required by the interior-point algorithm, we use an iterative method applied to the preconditioned system $\tilde{K}^{-1}Kx = \tilde{K}^{-1}b$. In particular, for each matrix-vector multiplication within the iterative method we require one solve operation by using the implicit factorization of $\tilde{K}$, as shown in Algorithm 2.

**Algorithm 1:** Factorize

**Data:** Augmented system matrix of form (16), parallel processes $\mathcal{P} = \{p_1, p_2, ..., p_N\}$

**Result:** Implicit factorization of perturbed augmented system matrix: $(\tilde{L}_0, \tilde{D}_0)$, $(L_i, D_i)_{i=1,...,N}$

**foreach** $p_i \in \mathcal{P}$ **do** — in parallel
  Factorize $L_i D_i L_i^T = K_i$
  Compute $B_i^T K_i^{-1} B_i$
**end**
**foreach** $(p_i, p_{i+1}) \in \mathcal{P} \times \mathcal{P}$ **do** — in parallel
  Compute $E_i, H_i, M_i$
**end**
**foreach** $p_i \in \mathcal{P} \setminus \{p_N\}$ **do** — in parallel
  $(\tilde{E}_i, \tilde{H}_i, \tilde{M}_i) :=$ SPARSIFY$(E_i, H_i, M_i)$
**end**
Sum $M_0$ up on process $p_1$ — collective operation
Collect $(\tilde{E}_i, \tilde{H}_i, \tilde{M}_i)_{i=1,...,N-1}$ on process $p_1$
Process $p_1$: $\tilde{M}_0 :=$ SPARSIFY$(M_0)$
Process $p_1$: Form $\tilde{C}$ from $(\tilde{E}_i, \tilde{H}_i, \tilde{M}_i)_{i=1,...,N-1}$, $\tilde{M}_0$
Process $p_1$: Factorize $\tilde{L}_0 \tilde{D}_0 \tilde{L}_0^T = \tilde{C}$

The iterative method used in our implementation is detailed in Section 4.1.

# 4 Implementation

As already mentioned, the implementations for this article are built on the parallel interior-point solver PIPS-IPM [22]. The parallelization mostly relies on MPI—shared memory parallelization, via OpenMP [10], is only used within the external linear solvers. In this article we use the sparse direct solver PARDISO 6.0[6] both for the computation of the local Schur complements (described in [23] and [22]), and the factorization and solution of the preconditioning matrix $\tilde{C}$.

[6] https://www.pardiso-project.org/

**Algorithm 2:** Solve

**Data:** Right hand side $b = (b_0, b_1, ..., b_N)$, parallel processes $\mathcal{P} = \{p_1, p_2, ..., p_N\}$, implicit factorization of perturbed augmented system matrix: $(\tilde{L}_0, \tilde{D}_0)$, $(L_i, D_i)_{i=1,...,N}$

**Result:** Solution $\Delta z = (\Delta z_0, \Delta z_1, ..., \Delta z_N)$

**foreach** $p_i \in \mathcal{P}$ **do** } in parallel
  Compute $q_i := K_i^{-1} b_i = (L_i^T)^{-1} D_i^{-1} L_i^{-1} b_i$
  Compute $w_i := B_i^T q_i$
**end**

Form $w_0 := b_0 - \sum_{i=1}^{N} w_i$ on process $p_1$ } collective operation

Process $p_1$: Compute $\Delta z_0 := (\tilde{L}_0^T)^{-1} \tilde{D}_0^{-1} \tilde{L}_0^{-1} w_0$

**foreach** $p_i \in \mathcal{P}$ **do** } in parallel
  Compute $v_i := B_i \Delta z_0$
  Compute $\Delta z_i := (L_i^T)^{-1} D_i^{-1} L_i^{-1} (b_i - v_i)$
**end**

However, if the number of zeroes in the Schur complement is expected to be small (which is estimated by a more general version of (41)), and the size of the Schur complement is sufficiently small, a dense direct solver is used and $\tilde{C}$ is not applied. Still, the Schur complement is sufficiently sparse for all LPs considered in this article and thus only PARDISO 6.0 is used. Also, interfaces to the sparse direct solvers MUMPS[7] and Intel MKL PARDISO[8] have been implemented, but are not used in this article. This section also describes newly implemented algorithms that are not directly related to the Schur complement decomposition—but which are still essential for obtaining a competitive and robust performance. Also quantitatively, our implementation constitutes a major extension of PIPS-IPM, with more than 50 000 lines of code having been added or modified. Thus in the remainder of this article we will refer to our extension of PIPS-IPM as *PIPS-IPM++*, or simply *PIPS++* for short.

## 4.1 Interior-point algorithm

Most of the algorithmic efforts in this article so far have concentrated on efficiently solving the linear system arising from an interior-point algorithm. However, of course also the interior-point algorithm itself is of high importance—to achieve fast and robust convergence. Even more so since energy system models are often numerically challenging (e.g., due to badly conditioned linear systems) already for commercial LP solvers; ELMOD being no exception. Thus the primary aim of the implementations described in the following are robustness, even at the cost of some performance. In the same way, we use more conservative parameters, e.g., for step lengths, than other open-source interior-point solvers.

PIPS-IPM uses the classic predictor-corrector method from Mehrotra [21]. To improve robustness, we have replaced this method by the multiple centrality correctors scheme of Colombo and Gondzio [8]. Moreover, since PIPS-IPM has been developed to solve quadratic problems (which have a tighter coupling

[7] http://mumps.enseeiht.fr/index.php?page=home

[8] https://software.intel.com/en-us/mkl-developer-reference-fortran-intel-mkl-pardiso-parallel-direct-sparse-solver-interface

between primal and dual variables than LPs in the KKT system), the same step length for primal and dual variables is used. As this article is concerned with LPs, different primal and dual step lengths, see e.g. [32], have been implemented to achieve faster convergence.

PIPS-IPM includes the Krylov subspace method BiCGSTAB [27] for the (preconditioned) iterative solution of the augmented system. We also use this implementation, but with some minor changes, such as a checks for divergence and stagnation. Also, we set a tight iteration limit of 75. The performance of preconditioned Krylov subspace methods is usually highly dependent on the preconditioner that is used. Therefore, especially the choice of the parameter $t$ in the construction of matrix $\tilde{C}$ (44) has a major impact. We observed good results with the following simple scheme: We initially set $t$ to $10^{-3}$ and whenever the BiCGSTAB method does not achieve sufficiently fast convergence we reduce the value of $t$—but never below $10^{-6}$. In the computational experiments for this article usually more than 95% of the non-zeroes in the Schur complement were removed by this method.

One can readily think of more advanced methods, such as precomputing the number of non-zeroes of $\tilde{C}$ before building it and also taking this value into account. Also, one could increase the value of $t$ again if the BiCGSTAB method converges quickly in several consecutive interior-point iterations. However, since the computational results of the simple scheme described above were already satisfactory, we did not explore any such more intricate methods.

## 4.2 Scaling and presolving

In this section, LPs are considered in a slightly different form than (8), namely with explicit variable bounds:

$$\min\{c^T x : Ax = b,\ \ell \leq x \leq u\}. \tag{51}$$

This notation allows for a better demonstration of the following scaling and presolving techniques.

Scaling is widely used in linear programming solvers to improve the condition of LPs [17, 24]. Scaling of (51) can be described by means of two diagonal matrices $R = (r_{i,j})$ and $C = (c_{i,j})$ with positive diagonal elements. The diagonals correspond to the row and column scaling factors respectively. Defining

$$\tilde{A} = RAC, \quad \tilde{b} = Rb, \quad \tilde{c} = Cc, \quad \tilde{\ell} = C^{-1}\ell, \text{ and } \tilde{u} = C^{-1}u$$

one obtains the scaled linear program

$$\min\{\tilde{c}^T x : \tilde{A}x = \tilde{b},\ \tilde{\ell} \leq x \leq \tilde{u}\}. \tag{52}$$

Each solution $\tilde{x}$ to (52) corresponds to a solution $x = C\tilde{x}$ to (51) with the same objective value. Note that scaling can be performed repetitively. A common objective of scaling methods is to reduce the ratio of the (absolute) smallest and largest entry in each row and column. For this article we have implemented geometric scaling, equilibrium scaling, and a combination of both [11]. While equilibrium scaling divides all coefficients in each nonzero row and column of the constraint matrix by the absolute largest entry within this vector, geometric scaling uses a simplified geometric mean of the absolute vector entries as divisor: For each column $A_j$ of the constraint matrix $A$ the divisor is

$\sqrt{\max_{i:a_{ij}\neq 0} |a_{ij}| \cdot \min_{i:a_{ij}\neq 0} |a_{ij}|}$, for each row $a_{i\cdot}$ it is
$\sqrt{\max_{j:a_{ij}\neq 0} |a_{ij}| \cdot \min_{j:a_{ij}\neq 0} |a_{ij}|}$. Geometric scaling is computationally more expensive than equilibrium scaling, since it is applied iteratively (up to 10 times in our implementation); equilibrium scaling on the other hand always converges in one step. However, the arrowhead structure allows us to perform all scaling methods efficiently in parallel. Therefore, the scaling run times are neglectable, and geometric scaling is used as default.

Another important ingredient of state-of-the-art linear programming solvers is presolving [1]. LPs resulting from a modeled application often contain redundant information. Thus it is often advisable to apply presolving routines to the LP in order to eliminate redundant parts. Presolving techniques for LPs aim to reduce the number of variables, constraints, and non-zeroes. Their major goal is to reduce the run time of the actual solver, but also the condition of the LP can be improved. Due to the distributed storage of the problem data, implementing presolving methods in the setting of this article is more complicated than for a customary solver. Thus, so far only the following, well-known [1, 14], methods have been implemented

- bound strengthening,
- singleton row presolving,
- parallel and nearly parallel row reduction.

Bound strengthening aims to find tighter variable bounds that are implicitly given in the constraints. Singleton row presolving applies to constraints that contain exactly one nonzero coefficients. Finally, parallel row reductions looks for constrains that are multiples of other ones (nearly parallel row reductions are a simple extension). All three methods have been implemented to run in distributed parallel.

## 4.3 Communicating the structure

Automatic detection of block structures in models can be challenging [12], hence, a processable block structure information based on the user's understanding of the model is often preferable. Note that there is no unique block structure in a model, but that there are many of them, depending on how rows and columns of the constraint matrix are permuted. The challenge is not mainly to find an annotation that is correct in the mathematical sense, but to find one where the decomposition approach described in this article is exploited best. A desirable annotation would provide a block structure with many independent blocks of similar size while the set of linking variables and linking constraints is small, most importantly the global ones. Both ELMOD and SIMPLE rely on the modeling language GAMS, which is widely used in the energy system modeling community. For communicating the LP together with its arrowhead structure, a callback interface between GAMS and PIPS++ has been implemented. Additional substructures (such local constraints and variables) are detected and exploited automatically by the solver—the user does not have to provide any information about them. More detail on the model annotation with GAMS can be found in [6]

Currently, model generation is a sequential process where GAMS generates one constraint after the other. Usually, model generation is fast and the time

consumption is negligible compared to the time consumed to solve the actual problem. However, with the specialized distributed parallel solver described in this article—which takes only minutes even for problems with more than a hundred million variables and constraints—the model generation time becomes relevant. Hence, it is worthwhile to mention that the previously introduced annotation can also serve as a basis to generate the model in a distributed fashion. Instead of generating one large monolithic model, the individual model blocks can be generated in parallel to exploit the power of distributed-memory architectures already during model generation. This approach is to be implemented within the near future.

For solving ELMOD instances, the model is annotated, which includes assigning each variable and constraint to a certain block. As detailed in Section 2.2, the vast majority of constraints and all but very few variables can be assigned to the diagonal blocks. The exceptions are global variables and all linking constraints, most of which are local linking constraints.

A dynamic implementation of this annotation realizes the automatic assignments. The total number of time steps and the time steps per diagonal block are predefined by the user. The total number of diagonal blocks is computed and variables and constraints are assigned.

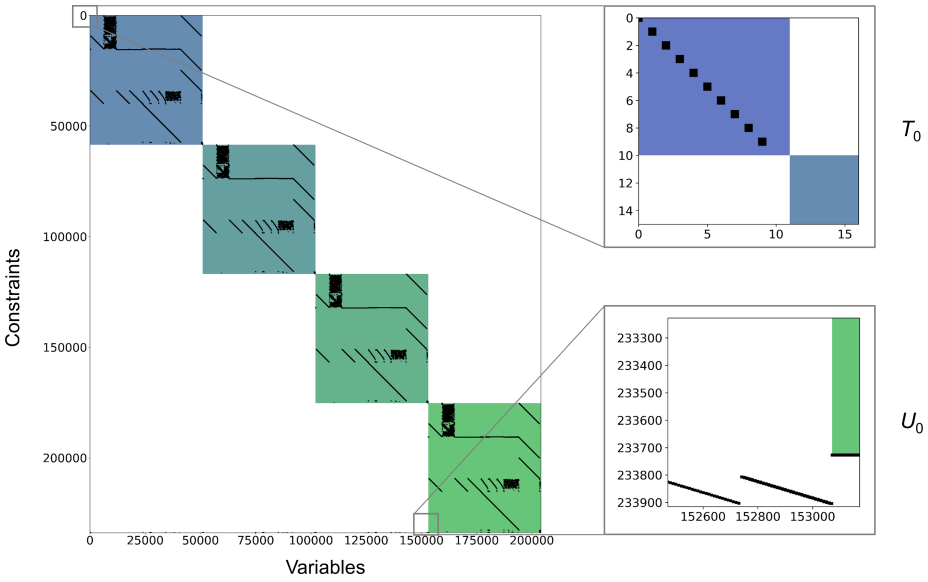


Figure 1: Constraint matrix of small model instance, encompassing Germany and a time frame of 40 hours.

Figure 1 displays the ensuing sorted structure of an exemplary small model instance. The model includes a physical grid representation of Germany with aggregated grids of its neighbors. A total of 40 hours are included, with ten hours for each parallel block. The linking constraints are displayed on the very bottom. The global variables are located on the left side of the structure. Upper limits for these global variables appear on the very top of the structure.

# 5 Computational results

Most of the computational experiments were performed on the supercomputer *JUWELS* at the Jülich Supercomputer Centre JUWELS is equipped with 2271 standard compute nodes (Dual Intel Xeon Platinum 8168), each with 96GB memory and 2x24 cores at 2.7 GHz. JUWELS also includes 240 large memory compute nodes with 192GB each (and otherwise the same configuration). The nodes are connected via a Mellanox EDR InfiniBand high-speed network. Only the standard compute nodes were used for PIPS++.

For comparison we use the state-of-the-art (commercial) LP solvers CPLEX 12.8[9], Gurobi 8.1[10], MOSEK 8.1[11], and Xpress 8.4.7[12]. We always use the interior-point algorithms of the respective solvers, since those achieve far better results for the instances at hand than default LP solving, or the simplex method. Also, crossover (which can considerably increase the run time) is turned off, as this feature is not implemented in our solver. If not mentioned otherwise, 16 threads are used for the commercial solvers, since this number seems to be the overall best one in terms of run time (with more threads usually a performance degradation occurs). Note that none of the commercial solvers allow for distributed parallel solution of linear programs. The results of the commercial solvers are given in anonymous form since these results are only supposed to serve as a comparison with our method. Still, it is worth noticing that there is no consistent ranking in terms of run-time among the commercial solvers for the instances considered in the following. On the contrary, one observes a considerable variation; the solver that performs best for the smallest instance is for example not able to solve any of the largest instances.

To demonstrate the scalability of PIPS++, a (relatively) small-scale SIMPLE instance with 5.6 million variables, 5.1 million constraints, 20.4 million non-zeroes, and 512 diagonal blocks is used. This instance can be handled with a single MPI process in reasonable time by PIPS++. Also, the instance allows for a customary power of 2 scaling plot. To achieve a good load-balancing, the number of MPI processes should divide the number of blocks—in this way each MPI process is assigned the same number of blocks. The scaling behavior of the commercial LP solvers and PIPS++ (denoted by *New solver*) is shown in Figure 2. Furthermore, the figure shows the scaling behavior of a reduced version (denoted by *New solver reduced*) of PIPS++ that just uses the Schur complement decomposition described in Section 3.1, but none of the additionally implemented algorithms described in Sections 3.2, 3.3, 4.1, and 4.2. For each MPI configuration of the new solver (including the reduced version) 2 OpenMP threads were used.

Figure 2 shows that the commercial solvers using one thread are considerably faster than PIPS++ using one MPI process (and two OpenMP threads)—by a factor of 5 or more. However, the commercial solvers do not exhibit a good scaling behavior. Three of the commercial solvers show a moderate speed-up up until 12 threads, but with more threads the run-time even deteriorates. PIPS++ on the other hand scales well, and requires merely 23 seconds with the maximum number of 512 MPI processes—whereas no commercial solver achieves a

[9] https://www.ibm.com/products/ilog-cplex-optimization-studio
[10] https://www.gurobi.com
[11] https://www.mosek.com/
[12] https://www.fico.com/en/products/fico-xpress-optimization

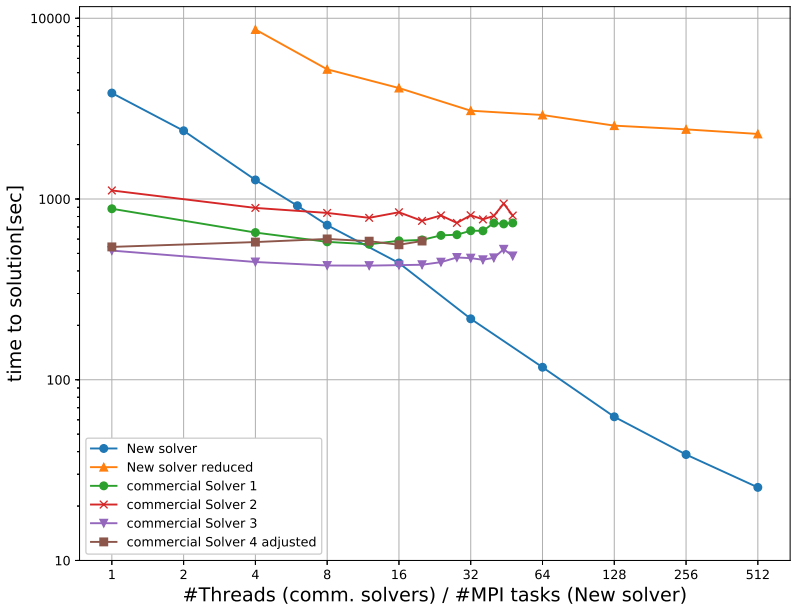


Figure 2: Scaling results of leading commercial LP solvers and PIPS++ on a SIMPLE instance. PIPS++ was run with 2 OpenMP threads per MPI process.

run-time below 400 seconds. Furthermore, the results of the reduced version of PIPS++ demonstrate that the straightforward application of the Schur complement decomposition without any additional techniques is not competitive. Another notable observation is that PIPS++ is for this instance already on a shared-memory architecture highly competitive with the commercial solvers: On 32 cores (16 MPI processes, with two OpenMP threads each), PIPS++ is as fast as any commercial solver—even if one looks at their best performance on 1 to 48 cores.

To demonstrate the performance of PIPS++ on large-scale instances, two different model configurations were utilized that vary in their grid configurations. The modeling of power plants was always block-wise, as opposed to aggregation by power plants, and both configurations encompass 8760 hours. The first configuration comprises the entire transmission grid of the Central Western European region which includes Germany, Austria, France, Belgium, Netherlands and Luxembourg. This configuration was further extended to a set of 19 European countries with detailed transmission grid representation. Neighboring countries were modeled as aggregated nodes. Both configurations are generated three times, based on the transmission and generation infrastructure of the years 2014, 2015 and 2016. The configuration with resulting instances corresponding to the Central Western European region are named *ELMOD_CWE*[13]

[13]Note that in a previous version of this technical report the ELMOD_CWE were not generated correctly and have thus been exchanged. Therefore, the sizes of the instances as

and the instances from the extended configuration corresponding to the 19 European countries are called *ELMOD_EU*. Notably, all ELMOD_EU instances have more than 300 000 linking constraints—but only a few (global) linking variables. Furthermore, two large-scale SIMPLE instances are used, which include considerably more time steps than the SIMPLE instance used for the scaling experiment above. In the following, these two instances are named *SIMPLE2* and *SIMPLE3*.

In each run PIPS++ uses the maximum number of MPI processes (equal to the number to blocks). Also, the PIPS++ uses two OpenMP threads for each MPI process. For the CWE instances the commercial solvers were run on the large memory nodes of JUWELS, because the 96 GB of memory available on the standard compute nodes were not sufficient, which caused all commercial solvers to abort. For all other instances the commercial solvers ran out of memory even on the large memory nodes of JUWELS. Thus, for those instances the commercial solvers were run on a large shared-memory machine at Zuse Institute Berlin[14] (ZIB) with the following specifications: Intel Xeon CPU E7-8880 v4 2.20GHz processor, 88 cores, and 2 TB memory. However, Solver 3, which actually performed best in the scaling experiment, crashed for several of the large-scale instances (and was always among the two slowest solvers for the remaining ones). Thus, it has been excluded from the presented results and only the other three commercial solvers are considered. The results on these large-scale instances are presented in Table 1. The first column specifies the instance name, and the next four columns provide the size of this instance, where *Blocks* specifies the number of diagonal blocks of the constraint matrix. The last four columns give the run time in seconds for PIPS++, as well as for the three commercial solvers. The $\star$ symbol marks that the corresponding instance could not be solved within the optimality tolerances, but still with a relatively small primal-dual gap: not more than 1%. *NO*, short for non-optimal, specifies that the respective solver could not solve the instance within acceptable tolerances—indeed the primal-dual gap on all results marked by *NO* was more than 100%. *TL* signifies that the (hard) time limit of 24 hours on JUWELS was hit.

Table 1: Computational results for large-scale instances

| | Size | | | | Run time (seconds) | | | |
|---|---|---|---|---|---|---|---|---|
| Instance | Variables | Constraints | Non-zeroes | Blocks | PIPS++ | Solver 1 | Solver 2 | Solver 4 |
| SIMPLE2 | 227 060 381 | 206 036 266 | 818 449 005 | 1024 | 546 | 38 800 | 69 377$^\star$ | NO |
| SIMPLE3 | 1 150 014 619 | 1 044 894 025 | 4 174 953 472 | 1024 | 13 170 | – | – | – |
| ELMOD_CWE14 | 85 585 234 | 98 532 392 | 271 621 021 | 438 | 239 | 6 181$^\star$ | 3 937 | 21 442 |
| ELMOD_CWE15 | 85 646 554 | 98 646 274 | 271 875 064 | 438 | 181 | 6 321$^\star$ | 6 245 | TL |
| ELMOD_CWE16 | 85 883 074 | 98 909 074 | 272 602 144 | 438 | 216 | 6 984$^\star$ | 5 190 | 67 941 |
| ELMOD_EU14 | 223 898 044 | 253 201 191 | 709 588 006 | 876 | 1 220$^\star$ | NO | 66 105 | NO |
| ELMOD_EU15 | 224 677 686 | 254 304 961 | 712 452 541 | 876 | 1 245$^\star$ | NO | 83 715 | NO |
| ELMOD_EU16 | 226 061 766 | 256 284 723 | 717 436 984 | 876 | 1 119$^\star$ | NO | 79 094 | NO |

Despite its large size, the SIMPLE2 instance can be solved within less than 10 minutes by PIPS++ (with 64 compute nodes and 3072 physical cores), compared to more than 10 hours for the best commercial solver. However, one should keep in mind that different machines needed to be used for this instance. The next instance SIMPLE3 is about five times larger, with more than four billion non-

well as the run times of the solvers differ compared to the previous version of this report

[14] www.zib.de

zeroes. Notably, this instances also has more than 100 000 linking variables and constraints. Still PIPS++ is able to solve the instance within four hours (with 128 compute nodes). Due to its huge size, this instance could not not be generated my GAMS as a single LP file, thus we could not perform comparisons with the commercial solvers.

The ELMOD CWE instances can be solved very quickly by PIPS++. However, also the best two commercial solvers can solve these instances relatively fast, in two hours or less (on the same architecture as PIPS++). Still, the speed-up of PIPS++ is significant. In particular, since far fewer MPI processes than for the SIMPLE instances are used.

Finally, all ELMOD EU instances are solved by PIPS++ in around 20 minutes. However, the instances can only be solved with a primal-dual gap (although considerably less than 1%). Interestingly, tests on the large-memory machine at ZIB revealed that this gap is not a result of our Schur complement preconditioning: The gap stays almost the same if the preconditioning approach is deactivated. Only one of the four commercial solver manages to solve the EU instances (and only on the large-memory machine). The run-times are between 18 and 24 hours, while PIPS++ takes only around 20 minutes. We also tried more aggressive numerical setting for the commercial solvers (in one case even according to the advice of the vendor). However, while this allowed Solver 1 to solve the EU instances, in around 40 to 45 thousand seconds, the behavior of Solver 4 did not change much, and Solver 2 even deteriorated in terms of solution quality: only one instances could be solved within the optimality tolerances. No further tests of different parameter settings were performed due to limited availability of the large-memory machine.

The results show that PIPS++ can considerably outperform even the best-performing commercial LP solver—by a factor of up to 67. While the commercial solvers are sequentially considerably faster (which is unsurprising, considering the decades of development that has been invested into them), the key advantage of the new approach is scalability. Notably, for the CWE instances only 19 compute nodes and for EU instances only 38 compute nodes were used. Compared to the one compute node (which in some cases is even much larger) that is occupied by each commercial solver, the run time per computing resources is most of the times better than the respective best result of the commercial solvers. Notably, PIPS++ is on most instances considered in this article already on shared-memory machines (both on JUWELS and the ZIB machine) faster than the commercial solvers—by a factor of up to 4. Thus, in the coming years PIPS++ might also considerably outperform the leading commercial solvers on standard desktop machines. Already, desktop machines with 32 cores are available[15], and machines with hundreds of cores are expected to become the standard in the near future.

# 6 Conclusion and outlook

This article has demonstrated how to exploit the power of distributed-memory supercomputers to drastically accelerate the solution time of complex linear energy system models as compared to the best commercial LP solvers. The key has been the development and implementation of scalable algorithms that

[15] https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-2990wx

can exploit the underlying structure of these problems. The algorithmic core is constituted by methods to efficiently solve the Schur complement system arising from the distributed decomposition approach used in this article. Still, to outperform the powerful state-of-the-art linear solvers, and to achieve robust convergence, also several generic algorithms such as a multiple-corrector scheme, scaling methods, and presolving techniques had to be implemented. The distributed storage of the problem data made this implementation a non-trivial endeavor, but also set the stage for efficient parallelization, and allowed to overcome memory limitations.

While the work described in this article already yields notable computational improvements, there are several paths for further development. A natural one is the implementation of additional presolving methods. Promising candidates are substitution of variables and elimination of linearly dependent rows. To improve robustness, also more aggressive scaling methods could be implemented. For the same reason, we plan to implement the homogeneous self-dual interior-point method [28]. Yet another extension, that is currently being implemented, is a hierarchical approach, which splits the Schur complement decomposition (and thus also the Schur complement) in several layers—with the aim to handle energy system models with even stronger linkage than ELMOD or SIMPLE. Finally, the authors plan to make the newly developed PIPS++ solver publicly available and hope that this software will allow the energy system community to overcome computational bottlenecks in the solution of their models.

# 7 Acknowledgments

The authors would like to thank Svenja Uslu for implementing parts of presolving and scaling routines and to Michael Bussieck, Fred Fiand and Manuel Wetzel for providing the interface to GAMS and the SIMPLE model. Furthermore, the authors would like to thank Cosmin Petra for several helpful discussions on PIPS-IPM. Last, but not least, the authors thank Thomas Breuer for his support on running the experiments on JUWELS. This work was supported by the BMWi project *Realisierung von Beschleunigungsstrategien der anwendungsorientierten Mathematik und Informatik für optimierende Energiesystemmodelle - BEAM-ME* (fund number 03ET4023DE). The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC).

# References

[1] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 2019. accepted for publication 2018-08-31.

[2] Erling D. Andersen, Jacek Gondzio, Csaba Mészáros, and Xiaojie Xu. *Implementation of Interior-Point Methods for Large Scale Linear Programs*, pages 189–252. Springer US, Boston, MA, 1996.

[3] Jason Barnett, Jean-Paul Watson, and David L. Woodruff. BBPH: using progressive hedging within branch and bound to solve multi-stage stochastic mixed integer programs. *Oper. Res. Lett.*, 45(1):34–39, 2017.

[4] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[5] C. W. Bomhof and H. A. van der Vorst. A parallel linear system solver for circuit simulation problems. *Numerical Linear Algebra with Applications*, 7(7-8):649–665, 2000.

[6] Thomas Breuer, Michael Bussieck, Karl-Kien Cao, Felix Cebulla, Frederik Fiand, Hans Christian Gils, Ambros Gleixner, Dmitry Khabi, Thorsten Koch, Daniel Rehfeldt, and Manuel Wetzel. Optimizing large-scale linear energy system problems with block diagonal structure by using parallel interior-point methods. In *Operations Research Proceedings 2017*, pages 641 – 647, 2018.

[7] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI Message Passing Interface Standard. In Karsten M. Decker and René M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218, Basel, 1994. Birkhäuser Basel.

[8] Marco Colombo and Jacek Gondzio. Further development of multiple centrality correctors for interior point methods. *Computational Optimization and Applications*, 41(3):277–305, Dec 2008.

[9] Marco Colombo, Jacek Gondzio, and Andreas Grothey. A warm-start approach for large-scale stochastic linear programs. *Math. Program.*, 127(2):371–397, 2011.

[10] B. R. de Supinski, T. R. W. Scogland, A. Duran, M. Klemm, S. M. Bellido, S. L. Olivier, C. Terboven, and T. G. Mattson. The Ongoing Evolution of OpenMP. *Proceedings of the IEEE*, 106(11):2004–2019, Nov 2018.

[11] Joseph M. Elble and Nikolaos V. Sahinidis. Scaling linear optimization problems prior to application of the simplex method. *Computational Optimization and Applications*, 52(2):345–371, 2012.

[12] Michael C. Ferris and Jeffrey D. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80(1):35–61, Jan 1998.

[13] Hans Christian Gils, Yvonne Scholz, Thomas Pregger, Diego Luca de Tena, and Dominik Heide. Integrated modelling of variable renewable energy-based power supply in Europe. *Energy*, 123:173–188, 2017.

[14] Jacek Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS Journal on Computing*, 9(1):73–91, 1997.

[15] Jacek Gondzio. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

[16] Jacek Gondzio and Robert Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, Jun 2003.

[17] Jonathan D. Hogg and Jennifer A. Scott. On the effects of scaling on the performance of Ipopt. *CoRR*, abs/1301.7283, 2013.

[18] Kibaek Kim, Cosmin G. Petra, and Victor M. Zavala. An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming. *SIAM Journal on Optimization*, 29(1):318–342, 2019.

[19] M. Lubin, C. G. Petra, M. Anitescu, and V. Zavala. Scalable stochastic optimization of complex energy systems. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, Nov 2011.

[20] Miles Lubin, R. Kipp Martin, Cosmin G. Petra, and Burhaneddin Sandikçi. On parallelizing dual decomposition in stochastic integer programming. *Oper. Res. Lett.*, 41(3):252–258, 2013.

[21] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM Journal on Optimization*, 2(3):435–449, 1992.

[22] Cosmin G. Petra, Olaf Schenk, and Mihai Anitescu. Real-time stochastic optimization of complex energy systems on high-performance computers. *Computing in Science & Engineering*, 16:32–42, 2014.

[23] Cosmin G. Petra, Olaf Schenk, Miles Lubin, and Klaus Gärtner. An Augmented Incomplete Factorization Approach for Computing the Schur Complement in Stochastic Optimization. *SIAM J. Scientific Computing*, 36(2), 2014.

[24] Nikolaos Ploskas and Nikolaos Samaras. The impact of scaling on simplex type algorithms. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 17–22, New York, NY, USA, 2013. ACM.

[25] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[26] Heinz Stigler and Christian Todem. Optimization of the austrian electricity sector (control zone of verbund apg) by nodal pricing. *Central European Journal of Operations Research*, 13(2):105, 2005.

[27] H. A. van der Vorst. BI-CGSTAB: A Fast and Smoothly Converging Variant of BI-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, March 1992.

[28] Robert J. Vanderbei. *The Homogeneous Self-Dual Method*, pages 361–381. Springer US, Boston, MA, 2008.

[29] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2014.

[30] A. J. Wathen. Preconditioning. *Acta Numerica*, 24:329–376, 2015.

[31] Frauke Wiese, Rasmus Bramstoft, Hardi Koduvere, Amalia Pizarro Alonso, Olexandr Balyk, Jon Gustav Kirkerud, Åsa Grytli Tveten, Torjus Folsland Bolkesjø, Marie Münster, and Hans Ravn. Balmorel open source energy system model. *Energy Strategy Reviews*, 20:26–34, 2018.

[32] Stephen J. Wright. *Primal-Dual Interior-Points Methods*. SIAM, Philadelphia, Pa, USA, 1997.

[33] Fuzhen Zhang. *The Schur Complement and its Applications*, volume 4 of *Numerical Methods and Algorithms*. Springer, New York, 2005.