Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

NILS–CHRISTIAN KEMPKE[1], DANIEL REHFELDT[2],
THORSTEN KOCH[3]

A Massively Parallel Interior-Point Method for Arrowhead Linear Programs with Local Linking Structure

[1] 0000-0003-4492-9818
[2] 0000-0002-2877-074X
[3] 0000-0002-1967-0077

Preprint

ZIB Report 24-13 (March 2026)

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 Method for Arrowhead Linear Programs with Local Linking Structure

Nils–Christian Kempke[*], Daniel Rehfeldt[†], Thorsten Koch[‡]

March 4, 2026

Abstract

In practice, non-specialized interior-point algorithms often cannot utilize the massively parallel compute resources offered by modern many- and multi-core compute platforms. However, efficient distributed solution techniques are required, especially for large-scale linear programs. This article describes a new decomposition technique for systems of linear equations, implemented in the parallel interior-point solver PIPS-IPM++. The algorithm exploits a matrix structure commonly found in optimization problems: a doubly bordered block-diagonal or arrowhead structure with linking constraints and variables often only linking few, consecutive blocks. This structure is preserved in the linear KKT systems solved during each iteration of the interior-point method. We present a hierarchical Schur complement decomposition that distributes and solves the linear optimization problem. It is designed for high-performance architectures and scales well with the availability of additional computing resources. The decomposition approach uses the border constraints' locality to decouple the factorization process. Our approach is motivated by large-scale economic dispatch problems but can also be applied to other problem classes. We demonstrate the performance of our method on a set of mid- to large-scale instances, some of which have more than 10^9 nonzeros in their constraint matrices.

1 Introduction

A recurring structural pattern in linear programs (LPs)—and more generally, in mixed-integer linear programs (MIPs)—is the *arrowhead* or *doubly bordered block-diagonal form*, as illustrated in fig. 1. Arrowhead LPs (AHLPs) contain two types of linking elements: *linking variables*, which vertically connect multiple diagonal blocks, and *linking constraints*, which connect blocks horizontally. This structure generalizes both primal and dual block-angular problems. In

[*] 0000-0003-4492-9818
[†] 0000-0002-2877-074X
[‡] 0000-0002-1967-0077

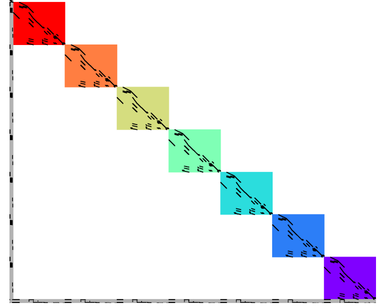


Figure 1: Constraint matrix with the arrowhead structure of a real-world ESM.

many AHLPs, linking variables and constraints exhibit *local structure*, connecting only a few—often two—consecutive blocks. This locality arises in a wide range of practical applications. Energy system models (ESMs), such as electricity market models with dispatch decisions [61], renewable expansion planning and dispatch [29, 40], and large-scale (stochastic) economic (re-)dispatch models, can often be decomposed spatially or temporally. Such decompositions naturally give rise to local linking constraints, for example through storage coupling or ramping limits. Local linking structure also arises in multi-stage LPs, where decisions at each stage depend on preceding stages. This includes multi-stage stochastic LPs, covering applications like asset–liability management, supply network design and supply chain planning, revenue management, and portfolio optimization [11, 16, 66]. Similarly, staircase LPs as used in production scheduling, inventory management, transportation, control, and design of multistage structures, exhibit local linking structure connecting consecutive diagonal blocks [23, 71]. Band-diagonal matrices arising in distribution planning can also be reformulated to expose arrowhead forms with local linkage [33]. While many optimization problems include integer variables (e.g., for unit commitment or network expansion planning), practitioners often prefer LP formulations because these scale well computationally and solving MIPs to optimality remains challenging for large instances, especially in the context of ESMs [39, 68, 70]. Even when a MIP is ultimately required, its LP relaxation remains critical, as many heuristic or decomposition-based algorithms rely on (near-)optimal LP solutions [5]. This structure is not limited to LPs and MIPs. Examples include nonlinear portfolio optimization [34], nonlinear dynamic optimization [72], model predictive control [59], nonlinear parameter estimation [78], and multi-stage nonlinear programs, such as the stochastic optimal power flow model in [56], all of which share the feature of localized linking structure.

Despite decades of progress in general-purpose LP solvers [6, 31], they often fail to solve large-scale AHLPs within reasonable time. With parallel hardware becoming increasingly available and affordable, one promising direction is the development of highly parallel, structure-exploiting algorithms. General LP solution techniques often scale poorly on such hardware, where specialized solutions promise to overcome scaling issues and push back the current computational limits by exploiting the power of high-performance computing (HPC). There has been much research activity in decomposing AHLPs—and nonlinear programs—and their subproblems. Solution techniques include specialized Simplex methods [23, 24], Danzig-Wolfe decomposition [71], Benders-

Decomposition [53, 79], Lagrangian decomposition [47], and frequently interior-point methods (IPMs) [9, 36, 49]. IPM solvers that specifically exploit this structure include: BlockIp [10], OOPS [35], PIPS-IPM [57], and MadNLP [56].

In IPMs, the focus of this work, often most of the algorithmic time is spent in solving (large-scale) sparse linear saddle-point systems for computing the Newton search direction of the method. This enables IPMs to employ specialized linear algebra routines directly working with the structure of a given problem class.

There are two main approaches for solving these systems: sparse direct factorizations ([18]) and iterative schemes with suitable preconditioners ([63]), leading to so-called inexact Newton methods. We refer the reader to [32] for an overview of the current state of the art of sparse direct and iterative solvers within IPMs. While factorization routines of sparse direct solvers can be slow and memory intensive for large-scale matrices, they often prove robust and exact enough to compute precise search directions. Additionally, direct factorizations readily support corrector step schemes ([54, 15]), solving the same linear system multiple times per iteration at little additional cost. High precision is especially required towards the end of IPMs, when the arising linear systems become increasingly ill-conditioned. During the earlier stages of an IPM, low accuracy solutions of the underlying linear systems can suffice [77]. Iterative schemes, often based on Krylov-subspace methods, are generally more lightweight in their computational effort and memory requirements but struggle to compute high accuracy solutions when the underlying matrix is ill-conditioned. They heavily rely on strong preconditioners which often need to be recomputed at every (other) iteration of the IPM. As saddle-point systems frequently appear in real-world applications their preconditioners have been studied extensively, also in the context of IPMs ([4, 60, 21, 45]). There also exist notable examples of preconditioners and iterative IPMs especially exploiting the AHLP structure. In PIPS-IPM and PIPS-IPM++ ([57, 61]) approximate decomposition methods are combined with Krylov subspace methods to solve large-scale AHLPs to high accuracy. In [50] the authors suggest several distributed preconditioners based on a distributed Schur complement method underlying an IPM algorithm. Also in the context of pure linear solvers much work has been dedicated to preconditioners exploiting AHLP matrix structures. In [62] the authors propose the ARMS linear solver applying a multilevel preconditioner based on incomplete LU factorization to a recursive nested-dissection ordering of their matrices (leading to nested arrowhead structures). In [75] the authors describe a multi-level Schur preconditioner (MLSR) using a top-down approach approximating the inverse of the original matrix which in [27] is further refined for indefinite systems, yielding GMLSR. A distributed implementation of GMLSR is discussed in [76]. Nevertheless, most commercial optimization software, including CPLEX [42], Gurobi [38], and the Cardinal Optimizer (COPT) [26], continues to rely on direct (exact) factorizations, valued for their stability and reliability across a wide range of problems.

In this work, we propose a hierarchical direct factorization scheme underlying an IPM. In previous work [61], we extended the solver PIPS-IPM, based on OOQP [28], from primal block-angular problems to AHLPs. The resulting solver PIPS-IPM++ could handle large-scale instances, however, its main bottleneck was the assembly and solution of the so-called (approximate) Schur complement. The size of the Schur complement is directly tied to the amount of linking

structure present in the AHLP. Since factorizing and solving with the Schur complement were performed sequentially or in shared-memory parallelism, this posed a bottleneck and limited the scalability of the algorithm.

While iterative or inexact hierarchical schemes based on approximate factorizations can appear attractive for reducing computational effort, our experience indicates that they often suffer from error propagation across recursive elimination levels. In IPMs, where the KKT systems become increasingly ill-conditioned as the iterates approach optimality, such accumulated inexactness can severely impair both numerical stability and the accuracy of the Newton search direction. Within the original PIPS-IPM++ framework, we observed that preconditioned BiCGStab could become unstable—even when the preconditioner was recomputed at each IPM iteration. For these reasons, we adopt a fully exact hierarchical direct factorization to ensure robustness and reproducibility, while maintaining parallel scalability through distributed elimination. Nonetheless, we see considerable potential for inexact hierarchical schemes as a complementary direction for future research.

The hierarchical ordering underlying our approach is conceptually related to orderings obtained through independent-set and nested-dissection strategies. In [64] and subsequently [62] the authors use *block independent sets* to split a matrix into arrowhead form and compute an incomplete LU factorization for this system. They then recursively apply their method to the obtained Schur complement block. In [75] the authors introduce HID, the *Hierarchical Interface Decomposition*, based on nested dissection (ND) which generates matrices that can be reordered into hierarchical arrowhead form (though their focus also lies on the Schur complement structure). Popular graph partitioning packages such as hMETIS [46], PaToH [12], and KaHyPar [22] based on multi-level hyper-graph partitioning can also be used to obtain similar matrix reorderings. The generic decomposition solver GCG [25] includes an arrowhead detection method based on hMETIS.

While these methods might be used to obtain an ordering generally suitable for HSCA, the main challenge lies in generating a well balanced block structure at each level of the algorithm. In HSCA, the arrowhead structure detection happens already on the LP system matrix rather than the KKT matrix. This allows us to employ a complex suite of structure preserving presolving techniques [30] (an extremely important part of solving an LP) to the LP formulation of our problem, without destroying the arrowhead structure or its well balancedness. Currently, HSCA relies on modeler-supplied decompositions that reflect the underlying physical or temporal structure, ensuring extremely well-balanced diagonal blocks and explicit control over the hierarchy of local links. However, we view automatic structure detection as an important part of a structure exploiting solver and are experimenting with detection mechanisms for large-scale matrices based on the KarHyPar package[1].

The main contribution of this article is the description and implementation of a novel approach within PIPS-IPM++, called the *hierarchical Schur complement approach* (HSCA). Exploiting the local structure of linking constraints, we recursively decompose the Schur complement into smaller, distributed components that can be solved in parallel. While our implementation specifically targets linking constraints connecting two consecutive blocks, most prevalent in

[1]https://gitlab.com/pips-ipmpp/detection-annotation

our models, constraints connecting multiple adjacent blocks can be treated analogously and the locality of linking variables can be similarly exploited. As IPMs also form the foundation for many quadratic and nonlinear solvers, the proposed method extends naturally to a wide class of structured nonlinear programs.

Our implementation is motivated by large-scale economic dispatch models with storage expansion. For the computational evaluation of HSCA, we rely on two different types of ESMs. First, we tested on the SIMPLE instances developed during the research projects BEAM-ME[2] and UNSEEN[3], see [61, 65]. These simplified models capture the essential features of ESMs, including all necessary DC power flow and storage balance constraints. The number and size of the diagonal blocks can be precisely controlled without incurring additional modeling overhead. Second, we consider a family of economic redispatch models generated with REMix-MISO, a variant of the REMix framework [29] developed at the German Aerospace Center[4]. These models are described in detail in [8, 69]. Using HSCA, we successfully solve instances with up to 1.7 million linking constraints, demonstrating substantial improvements in both scalability and performance.

2 Interior-point methods for linear programming

The two main methods for solving general LPs are the simplex algorithm [13] and IPMs [73]. Often, IPMs are more successful on large problems and offer more potential for parallelization. This success can be explained by the fact that the IPM's main computational effort is matrix factorization, and the amount of factorizations required grows relatively slowly with the problem size [58]. Many algorithms fall into the notion of IPMs, see [67, 73]. In this work, we use an infeasible primal-dual IPM similar to the ones described in [51, 54] using Mehrotra's predictor corrector scheme extended by Gondzio corrector steps [15]. In the following, we omit many details on the IPM method, and instead focus on the main feature of HSCA: the hierarchical decomposition of AHLPs. Details of the actual IPM can be found in [28, 48, 57] and the references mentioned therein.

2.1 Notation

We often write 0 for a zero-vector or matrix. As is often the case, we omit zero entries for large matrices with more complicated block structures. We sometimes selectively write 0 and omit entries, whenever it helps depict a given matrix block structure. I_n denotes the identity matrix of order n. When it is clear from the context, we omit the index and write I. The vector consisting of all ones, e.g., $(1, \dots, 1)^T \in \mathbb{R}^n$, will be denoted by e. For a vector $x \in \mathbb{R}^n$, $X = \operatorname{diag}(x) \in \mathbb{R}^{n \times n}$ denotes the diagonal matrix with x on its diagonal. For a given matrix $A \in \mathbb{R}^{m \times n}$ and two subsets $I \subset \{1, \dots, m\}$, $J \subset \{1, \dots, n\}$, we refer to the submatrix built from A by taking only the rows/columns indicated by I/J as $A_{I,\cdot}$ $/A_{\cdot,J}$ respectively.

[2] BEAM-ME project: http://www.beam-me-projekt.de/beam-me/EN/Home/home_node.html

[3] UNSEEN project: https://unseen-project.gitlab.io/home/

[4] German Aerospace Center: https://www.dlr.de/de

2.2 Linear programming, duality and optimality conditions

Consider the following LP in standard form

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0, \end{aligned}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$, as well as its associated dual formulation

$$\begin{aligned} \max_{y,z} \quad & b^T y \\ \text{subject to} \quad & A^T y + z = c \\ & z \geq 0, \end{aligned}$$

where $y \in \mathbb{R}^m, z \in \mathbb{R}^n$. We assume that all matrices considered here have full rank. It is well-known that if both problems are feasible, the optimal primal and dual objectives coincide [73]. The necessary KKT conditions to either the primal or the dual problem (which motivates the term primal-dual) are given by:

$$A^T y + z - c = 0 \tag{1}$$

$$Ax - b = 0 \tag{2}$$

$$ZXe = 0 \tag{3}$$

$$x, z \geq 0, \tag{4}$$

where $X = \operatorname{diag}(x)$ and $Z = \operatorname{diag}(z)$. All but the complementarity equation eq. (3) are linear.

2.3 Introduction to primal-dual interior-point methods

In primal-dual infeasible IPMs, perturbed KKT conditions are solved by Newton's method, namely eq. (3) is perturbed by $\tau > 0$:

$$ZXe = \tau e. \tag{5}$$

The so-called centrality factor τ is successively decreased, which guides the IPM along the *central path* defined by the values of τ close to the optimal solution until sufficient convergence has been reached. In doing so, the IPM maintains its iterates x, z strictly within the positive orthant.

We introduce the vector notation for eqs. (1), (2), (4) and (5) as well as an iteration index k. At each step k of the IPM, we approximately solve

$$\begin{bmatrix} A^T y^k + z^k \\ Ax^k \\ X^k Z^k e \end{bmatrix} = \begin{bmatrix} c \\ b \\ \tau^k e \end{bmatrix}. \tag{6}$$

The linear system that arises in each iteration l of Newton's method is

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z^{k,l} & 0 & X^{k,l} \end{bmatrix} \begin{bmatrix} \Delta x^{k,l} \\ \Delta y^{k,l} \\ \Delta z^{k,l} \end{bmatrix} = \begin{bmatrix} r_{x^{k,l}} \\ r_{y^{k,l}} \\ r_{z^{k,l}} \end{bmatrix},$$

where $r_{x^{k,l}}, r_{y^{k,l}}, r_{z^{k,l}}$ are residual right-hand sides. Recalling that $x^{k,l} > 0$, we eliminate the last row block

$$\Delta z^{k,l} = (X^{k,l})^{-1} r_{z^{k,l}} - (X^{k,l})^{-1} Z^{k,l} \Delta x^{k,l},$$

which produces the symmetric so-called *augmented system* [52, 74]

$$\begin{bmatrix} -(X^{k,l})^{-1} Z^{k,l} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x^{k,l} \\ \Delta y^{k,l} \end{bmatrix} = \begin{bmatrix} \hat{r}_{x^{k,l}} \\ r_{y^{k,l}} \end{bmatrix}. \tag{7}$$

In IPMs for LP, the number of inner iterations for approximately solving eq. (6) is generally kept to a single inner iteration l, after which the centrality factor τ is updated.

2.4 The Schur complement algorithm

Here, we recall the Schur complement algorithm, slightly modified for our purposes. We will then use it to derive HSCA. It can be thought of as a block Gaussian elimination. Consider the linear system

$$\begin{bmatrix} K & L \\ L^T & K_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_0 \end{bmatrix} \tag{8}$$

with matrices $K \in \mathbb{R}^{m \times m}$, $L \in \mathbb{R}^{m \times n}$, $K_0 \in \mathbb{R}^{n \times n}$, $n, m \in \mathbb{N}$ and vectors x_1, x_0, b_1, b_0 of appropriate dimensions. Given a factorization of K, we can compute and factorize the Schur complement $S := K_0 - L^T K^{-1} L$ of eq. (8) columnwise, as described in algorithm 1, and solve the linear system accordingly. The factorization of K is not required to be given as actual factors of a matrix factorization. Instead, any algorithm that allows solving linear equations with K suffices for the computation of S. We refer to such an algorithm as an *implicit factorization* of K.

Algorithm 1 Schur Complement Factorization

Input: System eq. (8)
1: Set $S = K_0$
2: **for** $i \in \{1, \ldots, n\}$ **do**
3: Solve $Kz = L_{.,i}$
4: Set $S_{.,i} = S_{.,i} - L^T z$
5: **end for**
6: Factorize S
7: **return**

3 A specialized parallel interior-point method

In the remainder of this section, we will show how PIPS-IPM++ employs a parallel Schur complement decomposition [61] to solve eq. (7) in a distributed fashion and in parallel. This forms the basis for the methods presented in section 4.

3.1 Message passing interface

We will be relying on the availability of communication between independent processes on a distributed computing system. Our implementation heavily uses the de facto standard in messaging protocols for distributed-memory computing, the message passing interface MPI [14]. MPI provides a set of collective and point-to-point communication routines for a given set of parallel processes. We commonly refer to these processes connected via MPI as *MPI processes* or just *processes*. Each of the MPI processes is assigned a unique global rank in $0, \ldots, N-1$, referred to as *global rank*, where N is the total number of processes. The choice of N as both the number of blocks and the number of MPI processes is intended, as they are essentially the same (as explained in the following chapters). The global ranks are associated with a global MPI communicator. Generally, communicators are used by processes for collective communication operations such as *Reduce* and *Allreduce*, where a specific reduction operation, e.g., summation, is used to combine the data of all MPI processes on respectively one or all processes. MPI offers the option of creating custom sub-communicators, each of which can contain an arbitrary subset of the N processes and assigns an additional (communicator-unique) rank to its processes. When discussing a process's rank, we mean its associated global rank (exceptions will be stated explicitly). Throughout this work, we use the Allreduce and Reduce operations to operate on vectors and matrices. Additionally, we use the concept of sub-communicators for the distribution of the linear algebra in HSCA.

3.2 A parallel Schur complement decomposition

The AHLPs whose system matrix is depicted in fig. 1 can mathematically be described as

$$
\begin{array}{rcccccccl}
\min & c_0^T x_0 & + & c_1^T x_1 & + & \cdots & + & c_N^T x_N & \\
\text{subject to} & A_0 x_0 & & & & & & & = b_0 \\
d_0 \leq & C_0 x_0 & & & & & & & \leq f_0 \\
& A_1 x_0 & + & B_1 x_1 & & & & & = b_1 \\
d_1 \leq & C_1 x_0 & + & D_1 x_1 & & & & & \leq f_1 \\
& \vdots & & & & \ddots & & & \vdots \\
& A_N x_0 & + & & & & + & B_N x_N & = b_N \\
d_N \leq & C_N x_0 & + & & & & + & D_N x_N & \leq f_N \\
& F_0 x_0 & + & F_1 x_1 & + & \cdots & + & F_N x_N & = b_{N+1} \\
d_{N+1} \leq & G_0 x_0 & + & G_1 x_1 & + & \cdots & + & G_N x_N & \leq f_{N+1} \\
& & & & & & & \ell_i \leq x_i \leq u_i & \forall i = 0, \ldots, N.
\end{array}
\tag{9}
$$

Here $x_i \in \mathbb{R}^{n_i}$ denote the decision variables, $c_i \in \mathbb{R}^{n_i}$ the objective vector and ℓ_i, $u_i \in \mathbb{R}^{n_i}$ the variable bounds. The vectors $b_i \in \mathbb{R}^{m_{i_A}}$, d_i, and $f_i \in \mathbb{R}^{m_{i_C}}$ denote the right-hand sides for equalities and lower and upper bounds for inequalities respectively. The system matrix is split into the sub-matrices $A_i \in \mathbb{R}^{m_{i_A} \times n_0}$, $B_i \in \mathbb{R}^{m_{i_A} \times n_i}$, $C_i \in \mathbb{R}^{m_{i_C} \times n_0}$, $D_i \in \mathbb{R}^{m_{i_C} \times n_0}$, $F_i \in \mathbb{R}^{m_{N+1_A} \times n_i}$,

$G_i \in \mathbb{R}^{m_{N+1_C} \times n_i}$ for $N \in \mathbb{N}$. We denote by $n_i \in \mathbb{N}$, the size of the i-th primal vector, by $m_{i_A} \in \mathbb{N}$ the row dimension of the i-th equality constraint sub-matrix A_i and by $m_{i_C} \in \mathbb{N}_0$ the row dimension of the i-th inequality constraint sub-matrix C_i. Using this notation, linking constraints are constraints associated with the blocks F_i and G_i, while linking variables are the variables in x_0. In the following, we will further distinguish linking constraints into *local linking constraints* and *global linking constraints*. Local linking constraints (also called two-links) only contain nonzeros in F_0 (or G_0) and two consecutive blocks F_i and F_{i+1} (or G_i and G_{i+1}) for exactly one $i \in \{1, \dots, N-1\}$. Global linking constraints constitute the rest of the linking constraints, in particular the rows that have entries in F_0, F_i, F_j (or G_0, G_i, G_j) where $|i-j| > 1$ and $i, j \in \{1, \dots, N\}$.

For clarity, we will omit the outer and inner iteration indices k, l introduced for IPMs in the rest of this section. Similarly, we will omit the C, D, and G matrices associated with inequality constraints of system eq. (9). Consequently, we refer to the sizes of the remaining A_i as $\mathbb{R}^{m_i \times n_0}$, B_i as $\mathbb{R}^{m_i \times n_i}$ and F_i as $\mathbb{R}^{m_{N+1} \times n_i}$, dropping the matrix index in m_{i_A}.

The augmented system eq. (7) corresponding to eq. (9) is

$$\begin{bmatrix} \Sigma_0 & & & & A_0^T & A_1^T & \dots & A_N^T & F_0^T \\ & \Sigma_1 & & & & B_1^T & & & F_1^T \\ & & \ddots & & & & \ddots & & \vdots \\ & & & \Sigma_N & & & & B_N^T & F_N^T \\ A_0 & & & & & & & & \\ A_1 & B_1 & & & & & & & \\ \vdots & & \ddots & & & & & & \\ A_N & & & B_N & & & & & \\ F_0 & F_1 & \dots & F_N & & & & & \end{bmatrix} \begin{bmatrix} \Delta x_0 \\ \Delta x_1 \\ \vdots \\ \Delta x_N \\ \Delta y_0 \\ \Delta y_1 \\ \vdots \\ \Delta y_N \\ \Delta y_{N+1} \end{bmatrix} = \begin{bmatrix} r_{x_0} \\ r_{x_1} \\ \vdots \\ r_{x_N} \\ r_{y_0} \\ r_{y_1} \\ \vdots \\ r_{y_N} \\ r_{y_{N+1}} \end{bmatrix},$$

where $\Sigma_i := -X_i^{-1} Z_i$ for $i = 0, \dots, N$. After a symmetric permutation, we get

$$\begin{bmatrix} K_1 & & & L_1 \\ & \ddots & & \vdots \\ & & K_N & L_N \\ L_1^T & \dots & L_N^T & K_0 \end{bmatrix} \begin{bmatrix} \Delta z_1 \\ \vdots \\ \Delta z_N \\ \Delta z_0 \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \\ b_0 \end{bmatrix}, \tag{10}$$

where

$$K_i := \begin{bmatrix} \Sigma_i & B_i^T \\ B_i & 0 \end{bmatrix}, \; K_0 := \begin{bmatrix} \Sigma_0 & A_0^T & F_0^T \\ A_0 & 0 & 0 \\ F_0 & 0 & 0 \end{bmatrix}, \; L_i := \begin{bmatrix} 0 & 0 & F_i^T \\ A_i & 0 & 0 \end{bmatrix},$$

$$\Delta z_0 := \begin{bmatrix} \Delta x_0 \\ \Delta y_0 \\ \Delta y_{N+1} \end{bmatrix}, \; \Delta z_i := \begin{bmatrix} \Delta x_i \\ \Delta y_i \end{bmatrix}, \; b_0 := \begin{bmatrix} r_{x_0} \\ r_{y_0} \\ r_{y_{N+1}} \end{bmatrix}, \; b_i := \begin{bmatrix} r_{x_i} \\ r_{y_i} \end{bmatrix},$$

for $i = \{1, \dots, N\}$. As described in [61], this system can be solved efficiently in parallel using a Schur complement decomposition.

As usual in direct methods, we distinguish between two phases, *factor* and *solve*. The factorization phase for solving system eq. (10) is described by algorithm 2. The solve phase for a given right-hand side is described by algorithm 3.

Algorithm 2 *Parallel* factorization

Input: System matrix eq. (10)

1: *Factorize (potentially implicit)* K_i for all $i \in \{1, \dots, N\}$
2: *Compute* $-L_i^T K_i^{-1} L_i$ for all $i \in \{1, \dots, N\}$
3: Sum Schur complement $S := K_0 - \sum_{i=1}^{N} L_i^T K_i^{-1} L_i$
4: Factorize S
5: **return**

Algorithm 3 *Parallel* solve

Input: Factorized system from algorithm 2 and right hand side in eq. (10)
Output: Solution $\Delta z_i,\ i \in \{0, \dots, N\}$

1: *Compute* $-L_i^T K_i^{-1} b_i$ for all $i \in \{1, \dots, N\}$
2: Sum Schur complement right hand side $\hat{b}_0 = b_0 - \sum_{i=1}^{N} L_i^T K_i^{-1} b_i$
3: Solve $S \Delta z_0 = \hat{b}_0$
4: *Compute modified right-hand sides* $\hat{b}_i = b_i - L_i \Delta z_0$
5: *Solve* $K_i \Delta z_i = \hat{b}_i$ for all $i \in \{1, \dots, N\}$
6: **return**

Algorithm 2 extends algorithm 1 in PIPS-IPM++. In its first implementation, PIPS-IPM++ used a combination of algorithm 1 and algorithm 2 to compute the Schur complement. Instead of forming K_i^{-1} explicitly, the Schur complement contributions $L_i^T K_i^{-1} L_i$ were computed by factoring K_i and solving $K_i z = l$ for each column l of L_i. Currently, the faster way of computing the blockwise Schur complement contributions $L_i K_i^{-1} L_i^T$ is the application of an incomplete (partial) LU factorization routine [57] to the extended system

$$\begin{bmatrix} K_i & L_i^T \\ L_i & 0 \end{bmatrix}.$$

By restricting pivoting to the $(1, 1)$ block and aborting after $m_i + n_i$ pivots, the transformed $(2, 2)$ block contains $-L_i^T K_i^{-1} L_i$. This approach is implemented in the solver PARDISO [1]. Similar to Algorithm 1 is it not necessary to compute actual factors for the factorization of K_i but it suffices to have an implicit factorization available.

lines 1 and 2 of algorithm 2, and lines 1, 4 and 5 of algorithm 3 can be executed in parallel and are highlighted in italic. When each MPI process has access to the data needed for the parallel steps, everything but the Schur complement factorization and the Schur complement solve can be run in distributed parallel. No process needs a representation of the whole problem: a process requires knowledge of only the system blocks i assigned to it and the 0 block as depicted in eq. (9). This also induces a limit on the number of processes and motivates the slightly ambiguous notation of N for the number of blocks and processes. A problem with N blocks can be distributed among at most N processes. We point out that line 3 in algorithm 2 and line 2 in algorithm 3 require communication between the processes to form the Schur complement and its right-hand side. To save memory and compute power, the Schur complement in PIPS-IPM++ is stored on a single process, rendering the communication in both steps a *reduce operation*. Processing the Schur complement on a single

MPI process requires additional communication (*broadcast*) of the solution z_0 in line 3 of algorithm 3.

3.3 Detecting structure in the linking constraints

One of the main contributions of [61] was detecting and exploiting additional structure in the linking part of the system matrix. This will also be essential in section 4. In the following, we summarize the most important findings.

We initially noticed that many linking constraints in ESMs were indeed local linking constraints (they link exactly two consecutive blocks in addition to the 0th block). This introduces additional structure to the problem: we can split F_i into local $\mathcal{F}_i$ and global $\mathbf{F}_i$ linking constraints, by defining

$$F_1 := \begin{bmatrix} \mathcal{F}_1 \\ 0 \\ \\ \vdots \\ \\ 0 \\ \mathbf{F}_1 \end{bmatrix}, F_2 := \begin{bmatrix} \mathcal{F}'_1 \\ \mathcal{F}_2 \\ 0 \\ \\ \vdots \\ \\ 0 \\ \mathbf{F}_2 \end{bmatrix}, F_3 := \begin{bmatrix} 0 \\ \mathcal{F}'_2 \\ \mathcal{F}_3 \\ 0 \\ \vdots \\ \\ 0 \\ \mathbf{F}_3 \end{bmatrix}, \ldots, F_{N-1} := \begin{bmatrix} 0 \\ \\ \vdots \\ \\ 0 \\ \mathcal{F}'_{N-2} \\ \mathcal{F}_{N-1} \\ \mathbf{F}_{N-1} \end{bmatrix}, F_N := \begin{bmatrix} 0 \\ \\ \vdots \\ \\ \\ 0 \\ \mathcal{F}'_{N-1} \\ \mathbf{F}_N \end{bmatrix},$$

with $\mathcal{F}_i \in \mathbb{R}^{l_i \times n_i}$, $\mathcal{F}'_i \in \mathbb{R}^{l_i \times n_{i+1}}$, $l_i \in \mathbb{N}_0$ for $i \in \{1, \ldots, N-1\}$ and $\mathbf{F}_i \in \mathbb{R}^{m_\mathbf{F} \times n_i}$ with $m_\mathbf{F} \in \mathbb{N}_0$. We set

$$\hat{F}_i := \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \mathcal{F}'_{i-1} \\ \mathcal{F}_i \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad F_0 := \begin{bmatrix} \mathcal{F}_{0,1} \\ \mathcal{F}_{0,2} \\ \\ \vdots \\ \\ \mathcal{F}_{0,N-2} \\ \mathcal{F}_{0,N-1} \\ \mathbf{F}_0 \end{bmatrix}, \quad \hat{F}_0 := \begin{bmatrix} \mathcal{F}_{0,1} \\ \mathcal{F}_{0,2} \\ \\ \vdots \\ \\ \mathcal{F}_{0,N-2} \\ \mathcal{F}_{0,N-1} \end{bmatrix},$$

with $\mathbf{F}_0 \in \mathbb{R}^{m_{\mathbf{F}_0} \times n_0}$, $\mathcal{F}_{0,i} \in \mathbb{R}^{l_i \times n_0}$, for $i \in \{1, \ldots, N-1\}$. Analogously expanding right hand side and variable vectors, the linear system eq. (10) becomes

$$\left[\begin{array}{ccccc|cccc} \Sigma_1 & B_1^T & & & & 0 & 0 & \hat{F}_1^T & \mathbf{F}_1^T \\ B_1 & 0 & & & & A_1 & 0 & 0 & 0 \\ & & \ddots & & & & \vdots & & \\ & & & \Sigma_N & B_N^T & 0 & 0 & \hat{F}_N^T & \mathbf{F}_N^T \\ & & & B_N & 0 & A_N & 0 & 0 & 0 \\ \hline 0 & A_1^T & & 0 & A_N^T & \Sigma_0 & A_0^T & \hat{F}_0^T & \mathbf{F}_0^T \\ 0 & 0 & \ldots & 0 & 0 & A_0 & 0 & 0 & 0 \\ \hat{F}_1 & 0 & & \hat{F}_N & 0 & \hat{F}_0 & 0 & 0 & 0 \\ \mathbf{F}_1 & 0 & & \mathbf{F}_N & 0 & \mathbf{F}_0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_N \\ \Delta y_N \\ \Delta x_0 \\ \Delta y_0 \\ \Delta y_{N+1} \\ \mathbf{\Delta y_{N+1}} \end{bmatrix} = \begin{bmatrix} r_{x_1} \\ r_{y_1} \\ \vdots \\ r_{x_N} \\ r_{y_N} \\ r_{x_0} \\ r_{y_0} \\ r_{y_{N+1}} \\ \mathbf{r_{y_{N+1}}} \end{bmatrix}. \quad (11)$$

We note that the locality of many of the linking constraints also induces a arrowhead block structure on the Schur complement of eq. (10). PIPS-IPM++ exploits this structure to use a sparse direct solver for the Schur complement. We refer the reader to appendix A for more details.

4 A hierarchical solution approach

As already mentioned in section 1, one of the main bottlenecks in PIPS-IPM++ for solving large problems or problems with many linking constraints is the size of the Schur complement. In the following, we outline an algorithm that circumvents this bottleneck by further exploiting the local linking constraints. It splits the Schur complement into multiple smaller Schur complements hierarchically connected in the factorization and solve processes.

4.1 The dense layer

After another symmetric permutation in the borders, which moves the local linking constraints $\hat{F}_i^T/\hat{F}_i$ $i = 0, \ldots, N$ to the front/top of their respective blocks, eq. (11) becomes

$$\left[\begin{array}{cccc|c|cccc} \Sigma_1 & B_1^T & & & & \hat{F}_1^T & 0 & 0 & \mathbf{F}_1^T \\ B_1 & 0 & & & & 0 & A_1 & 0 & 0 \\ & & \ddots & & & \vdots & & \vdots & \\ & & & \Sigma_N & B_N^T & \hat{F}_N^T & 0 & 0 & \mathbf{F}_N^T \\ & & & B_N & 0 & 0 & A_N & 0 & 0 \\ \hline \hat{F}_1 & 0 & \ldots & \hat{F}_N & 0 & 0 & \hat{F}_0 & 0 & 0 \\ \hline 0 & A_1^T & & 0 & A_N^T & \hat{F}_0^T & \Sigma_0 & A_0^T & \mathbf{F}_0^T \\ 0 & 0 & \ldots & 0 & 0 & 0 & A_0 & 0 & 0 \\ \mathbf{F}_1 & 0 & & \mathbf{F}_N & 0 & 0 & \mathbf{F}_0 & 0 & 0 \end{array}\right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_N \\ \Delta y_N \\ \Delta y_{N+1} \\ \Delta x_0 \\ \Delta y_0 \\ \Delta \mathbf{y_{N+1}} \end{bmatrix} = \begin{bmatrix} r_{x_1} \\ r_{y_1} \\ \vdots \\ r_{x_N} \\ r_{y_N} \\ r_{y_{N+1}} \\ r_{x_0} \\ r_{y_0} \\ \mathbf{r_{y_{N+1}}} \end{bmatrix}.$$

We can apply the Schur complement decomposition in algorithm 1 by setting

$$K^{\text{dense}} := \begin{bmatrix} \Sigma_1 & B_1^T & & & & \hat{F}_1^T \\ B_1 & 0 & & & & 0 \\ & & \ddots & & & \vdots \\ & & & \Sigma_N & B_N^T & \hat{F}_N^T \\ & & & B_N & 0 & 0 \\ \hat{F}_1 & 0 & \ldots & \hat{F}_N & 0 & 0 \end{bmatrix},\ K_0^{\text{dense}} := \begin{bmatrix} \Sigma_0 & A_0^T & \mathbf{F}_0^T \\ A_0 & 0 & 0 \\ \mathbf{F}_0 & 0 & 0 \end{bmatrix},\ L^{\text{dense}} := \begin{bmatrix} 0 & 0 & \mathbf{F}_1^T \\ A_1 & 0 & 0 \\ & \vdots & \\ 0 & 0 & \mathbf{F}_N^T \\ A_N & 0 & 0 \\ \hat{F}_0 & 0 & 0 \end{bmatrix}$$

in eq. (8). We call the Schur complement system obtained this way the *dense layer* of the multi-layered HSCA as we do not detect any additional structure in the Schur complement of this layer and thus factorize it with a dense direct linear solver. Given an implicit factorization of the resulting inner system K^{dense}, we can obtain an implicit factorization of the whole system. Using a Schur complement to mitigate the fill-in generated by dense columns in IPMs is usually applied when solving the *normal equations* [3, 55]. When solving the *augmented system*, dense columns are usually treated by adapting the choice of pivoting. The method presented here can be seen as a similar approach [52], removing both, dense rows and columns from the system.

4.2 The inner linear systems

To actually apply algorithm 1 to the dense layer, we must be able to solve the inner linear system

$$K^{\text{dense}} = \left[\begin{array}{cccccc|c} \Sigma_1 & B_1^T & & & & & \hat{F}_1^T \\ B_1 & 0 & & & & & 0 \\ & & \ddots & & & & \\ & & & & \Sigma_N & B_N^T & \hat{F}_N^T \\ & & & & B_N & 0 & 0 \\ \hline \hat{F}_1 & 0 & & & \hat{F}_N & 0 & 0 \end{array}\right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_N \\ \Delta y_N \\ \Delta y_{N+1} \end{bmatrix} = \begin{bmatrix} r_{x_1} \\ r_{y_1} \\ \vdots \\ r_{x_N} \\ r_{y_N} \\ r_{y_{N+1}} \end{bmatrix}. \tag{12}$$

Since eq. (12) has the same structure as eq. (10), we can solve it using the parallel factorization and solve for arrowhead systems introduced in algorithms 2 and 3. Computing an explicit factorization of the diagonal block matrices would result in a two layered HSCA system.

To distribute and parallelize the solution process further, we instead split the linear system eq. (12) and apply a recursive Schur complement decomposition by permuting subsets of the two-link constraints "into" eq. (12). We recall the structure of eq. (12) with the expanded linking constraints:

$$\left[\begin{array}{cccccccc|cccc} \Sigma_1 & B_1^T & & & & & & & \mathcal{F}_1^T & 0 & & \\ B_1 & 0 & & & & & & & 0 & 0 & & \\ & & \Sigma_2 & B_2^T & & & & & \mathcal{F}_1'^T & \mathcal{F}_2^T & & \\ & & B_2 & 0 & & & & & 0 & 0 & & \\ & & & & \ddots & & & & & & \ddots & \\ & & & & & & \Sigma_N & B_N^T & & & & \mathcal{F}_{N-1}'^T \\ & & & & & & B_N & 0 & & & & 0 \\ \hline \mathcal{F}_1 & 0 & \mathcal{F}_1' & 0 & & & & & & & & \\ & & \mathcal{F}_2 & 0 & & & & & & & & \\ & & & & \ddots & & & & & 0 & & \\ & & & & & \mathcal{F}_{N-1}' & 0 & & & & & \end{array}\right] \begin{bmatrix} \Delta x_1 \\ \Delta y_1 \\ \vdots \\ \Delta x_N \\ \Delta y_N \\ \Delta y_{N+1,1} \\ \vdots \\ \Delta y_{N+1,N-1} \end{bmatrix} = \begin{bmatrix} r_{x_1} \\ r_{y_1} \\ \vdots \\ r_{x_N} \\ r_{y_N} \\ r_{y_{N+1,1}} \\ \vdots \\ r_{y_{N+1,N-1}} \end{bmatrix}. \tag{13}$$

Next, we group consecutive diagonal blocks of eq. (13) into $k \in \mathbb{N}$ subsets by setting $i_1 = 1 < i_2 < \cdots < i_{k+1} = N+1$, $k \in \mathbb{N}$ and assigning to each subset $j \in \{1, \ldots, k\}$ the consecutive range of block indices $i_j, \ldots, i_{j+1}-1$. We define

$$\hat{K}_j := \begin{bmatrix} \Sigma_{i_j} & B_{i_j}^T & & & & & \\ B_{i_j} & 0 & & & & & \\ & & \Sigma_{i_j+1} & B_{i_j+1}^T & & & \\ & & B_{i_j+1} & 0 & & & \\ & & & & \ddots & & \\ & & & & & \Sigma_{i_{j+1}-1} & B_{i_{j+1}-1}^T \\ & & & & & B_{i_{j+1}-1} & 0 \end{bmatrix},$$

$$\Delta\hat{x}_j := \begin{bmatrix} \Delta x_{i_j} \\ \Delta y_{i_j} \\ \vdots \\ \Delta x_{i_{j+1}-1} \\ \Delta y_{i_{j+1}-1} \end{bmatrix}, \ \Delta\hat{y}_j := \begin{bmatrix} y_{N+1,i_j} \\ \vdots \\ y_{N+1,i_{j+1}-2} \end{bmatrix}, \ r_{\hat{x}_j} := \begin{bmatrix} r_{x_{i_j}} \\ r_{y_{i_j}} \\ \vdots \\ r_{x_{i_{j+1}-1}} \\ r_{y_{i_{j+1}-1}} \end{bmatrix}, \ r_{\hat{y}_j} := \begin{bmatrix} r_{y_{N+1,i_j}} \\ \vdots \\ r_{y_{N+1,i_{j+1}-2}} \end{bmatrix},$$

$$
\hat{\mathcal{F}}_j'^T := \begin{bmatrix} \mathcal{F}_{i_j}'^T \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \;
\hat{B}_j^T := \begin{bmatrix} \mathcal{F}_{i_j}^T & 0 & & \\ 0 & 0 & & \\ \mathcal{F}_{i_j}'^T & \mathcal{F}_{i_j+1}^T & & \\ & 0 & & \\ & & \ddots & \\ & & & \mathcal{F}_{i_{j+1}-2}'^T \\ & & & 0 \end{bmatrix}, \;
\hat{\mathcal{F}}_j^T := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ \mathcal{F}_{i_{j+1}-1}^T \\ 0 \end{bmatrix},
$$

for $j = 1, \ldots, k$ ($\hat{\mathcal{F}}_j'^T$, $\hat{\mathcal{F}}_j^T$ are only defined up to $k-1$). We can regroup eq. (13) into

$$
\left[\begin{array}{cccc|ccccc}
\hat{K}_1 & & & & \hat{B}_1^T & \hat{\mathcal{F}}_1^T & & & \\
& \hat{K}_2 & & & & \hat{\mathcal{F}}_1^{T'} & \hat{B}_2^T & \hat{\mathcal{F}}_2^T & \\
& & \ddots & & & & & \ddots & \\
& & & \hat{K}_k & & & & \hat{\mathcal{F}}_{k-1}^{T'} & \hat{B}_k^T \\
\hline
\hat{B}_1 & & & & & & & & \\
\hat{\mathcal{F}}_1 & \hat{\mathcal{F}}_1' & & & & & & & \\
& \hat{B}_2 & & & & & & & \\
& \hat{\mathcal{F}}_2 & \ddots & & & & 0 & & \\
& & \ddots & \hat{\mathcal{F}}_{k-1}' & & & & & \\
& & & \hat{B}_k & & & & &
\end{array}\right]
\begin{bmatrix} \Delta\hat{x}_1 \\ \Delta\hat{x}_2 \\ \vdots \\ \Delta\hat{x}_k \\ \Delta\hat{y}_1 \\ \Delta y_{N+1,i_2-1} \\ \Delta\hat{y}_2 \\ \vdots \\ \Delta y_{N+1,i_k-1} \\ \Delta\hat{y}_k \end{bmatrix}
=
\begin{bmatrix} r_{\hat{x}_1} \\ r_{\hat{x}_2} \\ \vdots \\ r_{\hat{x}_k} \\ r_{\hat{y}_1} \\ r_{y_{N+1,i_2-1}} \\ r_{\hat{y}_2} \\ \vdots \\ r_{y_{N+1,i_k-1}} \\ r_{\hat{y}_k} \end{bmatrix}.
\tag{14}
$$

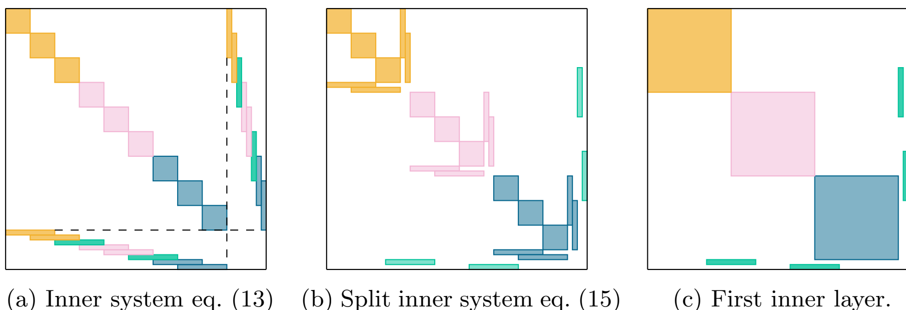
A symmetric permutation of eq. (14) gives

$$
\left[\begin{array}{cccccc|ccc}
\hat{K}_1 & \hat{B}_1^T & & & & & \hat{\mathcal{F}}_1^T & & \\
\hat{B}_1 & 0 & & & & & 0 & & \\
& & \hat{K}_2 & \hat{B}_2^T & & & \hat{\mathcal{F}}_1^{T'} & \hat{\mathcal{F}}_2^T & \\
& & \hat{B}_2 & 0 & & & 0 & 0 & \\
& & & & \ddots & & & & \ddots \\
& & & & \hat{K}_k & \hat{B}_k^T & & & \hat{\mathcal{F}}_{k-1}^{T'} \\
& & & & \hat{B}_k & 0 & & & 0 \\
\hline
\hat{\mathcal{F}}_1 & 0 & \hat{\mathcal{F}}_1' & 0 & & & & & \\
& & \hat{\mathcal{F}}_2 & 0 & & & & 0 & \\
& & & \ddots & \hat{\mathcal{F}}_{k-1}' & 0 & & &
\end{array}\right]
\begin{bmatrix} \Delta\hat{x}_1 \\ \Delta\hat{y}_1 \\ \vdots \\ \Delta\hat{x}_k \\ \Delta\hat{y}_k \\ \Delta y_{N+1,i_2-1} \\ \vdots \\ \Delta y_{N+1,i_k-1} \end{bmatrix}
=
\begin{bmatrix} r_{\hat{x}_1} \\ r_{\hat{y}_1} \\ \vdots \\ r_{\hat{x}_k} \\ r_{\hat{y}_k} \\ r_{y_{N+1,i_2-1}} \\ \vdots \\ r_{y_{N+1,i_k-1}} \end{bmatrix}.
\tag{15}
$$

By setting

$$
K_i^{inner} := \begin{bmatrix} \hat{K}_i & \hat{B}_i^T \\ \hat{B}_i & 0 \end{bmatrix}, \quad K_0^{inner} := 0, \quad L_i^{inner} := \begin{bmatrix} \ldots 0 & \hat{\mathcal{F}}_{i-1}^{T'} & \hat{\mathcal{F}}_i^T & 0 \ldots \\ \ldots 0 & 0 & 0 & 0 \ldots \end{bmatrix},
$$

the system in eq. (15) is of the same form as eq. (10), and we can use algorithm 2 and algorithm 3 to factorize and solve it in parallel. However, instead of computing a direct factorization of each diagonal 2×2-block, we notice their



(a) Inner system eq. (13) (b) Split inner system eq. (15) (c) First inner layer.

Figure 2: Permuting the inner linear system

arrowhead structure:

$$K_i^{inner} = \left[\begin{array}{ccccccc|cccc}
\Sigma_{i_j} & B_{i_j}^T & & & & & & \mathcal{F}_{i_j}^T & 0 & & \\
B_{i_j} & 0 & & & & & & 0 & 0 & & \\
 & & \Sigma_{i_j+1} & B_{i_j+1}^T & & & & \mathcal{F}_{i_j}^{T'} & \mathcal{F}_{i_j+1}^T & & \\
 & & B_{i_j+1} & 0 & & & & 0 & 0 & & \\
 & & & & \ddots & & & & & \ddots & \\
 & & & & & \Sigma_{i_{j+1}-1} & B_{i_{j+1}-1}^T & & & & \mathcal{F}_{i_{j+1}-2}^{T'} \\
 & & & & & B_{i_{j+1}-1} & 0 & & & & 0 \\
\hline
\mathcal{F}_{i_j} & 0 & \mathcal{F}_{i_j}^{'} & 0 & & & & & & & \\
0 & 0 & \mathcal{F}_{i_j+1} & 0 & & & & & & & \\
 & & & & \ddots & & & & 0 & & \\
 & & & & & \mathcal{F}_{i_{j+1}-2}^{'} & 0 & & & &
\end{array}\right], \tag{16}$$

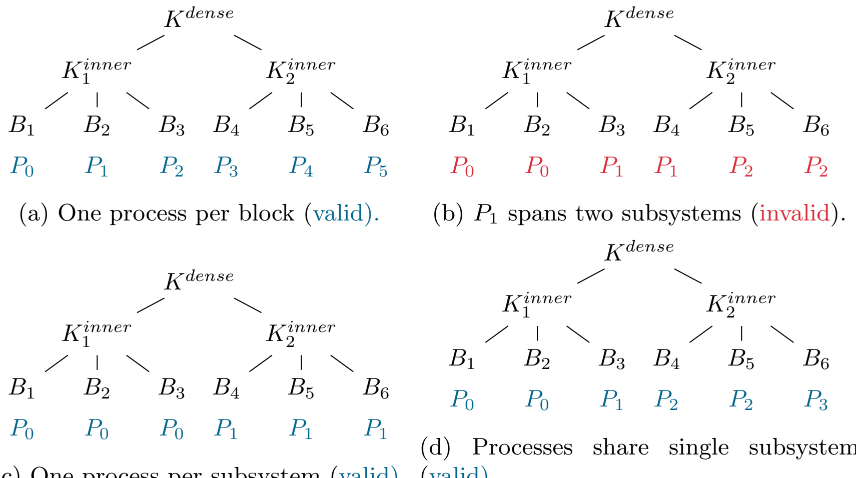
$i = 1, \ldots, k$. Each diagonal block itself is of the form eq. (13) and can again be handled by algorithms algorithm 2 and algorithm 3 to obtain an implicit factorization of K_i^{inner}. Alternatively, we can split each K_i^{inner} further by applying the same technique as outlined above, recursively.

We have decomposed the linear system eq. (12) into k smaller linear systems defined by the system matrices K_i^{inner}. We illustrate this process schematically in fig. 2. In fig. 2a, we are given a system K^{dense} with nine diagonal blocks and expanded two-link structure. We sub-divide this system into three smaller ones. For this purpose, we color-coded the components belonging to each subsystem with the same color. By permuting all local linking constraints but the ones linking the subsystems (depicted in green) further into the system, we obtain the system in fig. 2b, with arrowhead systems K_i^{inner}, $i = 1, 2, 3$, on the diagonal. Interpreting the system in fig. 2b as displayed in fig. 2c, we can use algorithm 2 for its factorization, assuming an implicit factorization for K_i^{inner} is available. When going from eq. (14) to eq. (15) it is not necessary that all linking constraints permuted *into* the system are two-links. For $\hat{\mathcal{F}}_j$ and $\hat{\mathcal{F}}_j'$, $j = 1, \ldots, k-1$, we can allow linking constraints connecting at most the diagonal blocks contained in the range $i_j, \ldots, i_{j+1} - 1$.

Recursively applying the splitting procedure to each K_i^{inner} would create another layer in the Schur complement decomposition and further shrink the lowest-level systems. Given enough blocks, this enables an arbitrary number of

layers. In practice, the recursive Schur complement decomposition affects the computational cost of a linear system solve and must be weighed against the benefit of smaller Schur complements and their distributed computation. We have not encountered systems or problems where more than four layers, one dense layer and three inner layers, were beneficial.

Note that the distribution of the linear systems and their subsystems naturally groups the MPI processes into subsets defined via the blocks assigned to the systems K_i^{inner}. We schematically display the assignment of different



(a) One process per block (valid).

(b) P_1 spans two subsystems (invalid).

(c) One process per subsystem (valid).

(d) Processes share single subsystem (valid).

Figure 3: System factorization/solve tree: valid and invalid process assignments.

numbers of processes to different subsystems in fig. 3. We display the factorization/solve tree of a two level HSCA system, splitting the inner system into two subsystems and omitting the dense layer. We denote the problem blocks representatively by B_i, $i = 1, \ldots, 6$. Assignment of a process $i = 0, \ldots, 5$ (MPI assigns processes ranks starting from 0) to a block is visualized by placing P_i below the respective block in the tree. In fig. 3a processes P_0, P_1, and P_2 are assigned to K_1^{inner} and P_3, P_4, and P_5 to K_2^{inner}. When splitting linear systems into smaller subsystems, the processes associated with the subsystem's blocks are grouped together and communally compute the implicit factorization of the subsystem. Within PIPS-IPM++, this grouping is realized using MPI groups and communicators, which simplifies the setup and the data transfer required for subsystem factorization and solve. This assignment becomes non-trivial when fewer than N processes are available. The parallelization of HSCA relies on the subsystems being factorized and used for solving linear equations in parallel. If a single process is responsible for factorizing two linear systems in the same level, it will have to communicate with the other processes responsible for each of the systems. This situation is depicted in fig. 3b, where P_1 is assigned to K_1^{inner} and K_2^{inner}. As MPI communication operations are blocking until all processes are available, this assignment blocks the parallel factorization of K_1^{inner} and K_2^{inner} and, at least partially, renders the computations sequential. Thus, an assignment of processes to blocks must consider the splitting within HSCA. As a result, in the current development, we have implemented a simple heuristic that enforces two rules: either one process is fully responsible for any

linear system of a given level it is assigned to, as in fig. 3c, or it participates in the processing of exactly one subsystem as in fig. 3d.

5 Computational experiments

In this section, we describe numerical results obtained with HSCA. All experiments with PIPS-IPM++ were conducted on the JUWELS supercomputer [44] at Forschungszentrum Jülich. JUWELS has 2,271 standard compute nodes (Dual Intel Xeon Platinum 8168), each with 96GB memory and 2x24 cores. The nodes are connected via a Mellanox EDR InfiniBand high-speed network. The experiments with other commercial solvers were conducted on a compute cluster (Intel Xeon Gold 6338) at the Zuse Institute Berlin, each with 1024GB memory and 2x32 cores. As this paper is not a commercial solver comparison, we have anonymized all results obtained with commercial optimization software. Our software is freely available on GitHub.

PIPS-IPM++ is able to not only leverage parallelism via MPI but also shared memory parallelism via OpenMP [17]. OpenMP is mainly used for the factorization and the solution of the linear Schur complement systems within the parallel solver PARDISO (see [61]). In preliminary experiments with HSCA, we compared the performance of the linear solvers MA86 [41], MA57 [19], MA27 [20], MUMPS [2], PARDISO [1], MKL PARDISO [43], and WSMP [37]. Our initial studies focused on two key aspects: the speed of factorization for medium-size linear systems and strong scalability when solving a given linear system with multiple right-hand sides, which is the most computationally demanding part of HSCA. We finally settled on MA57. This deviates from the latest incomplete LU factorization approach applied in [57, 61] and PIPS-IPM++' original implementation. While the factorization routine of MA57 is slower than that of PARDISO, particularly for larger systems, it allows for more efficient parallel solves. Specifically, the sequential MA57 is thread-safe, enabling the use of OpenMP to solve multiple right-hand sides simultaneously by calling the solver from different threads using the same factorization. In our experience, this “outside” multi-threading of the solve routine scales much better than the solve routines of all other solvers we tested. Additionally, by not relying onPARDISO's parallelized incomplete LU factorization for computing the bottommost Schur complement contributions, we gain numerical precision on the lowest HSCA level, albeit at the cost of losing some of PARDISO's speed-up. PARDISO on the other hand showed greater robustness when solving rank-deficient linear systems. In summary, for medium-size matrices where repeated linear solves dominate the computation, the slower factorization of MA57 is outweighed by its superior scalability and precision in solving multiple right-hand sides. These advantages made MA57 the preferred solver for our setting.

The scaling behavior of the hierarchical approach

We first assess the scaling behavior of our implementation. In fig. 4 we show the behavior of our solver on the *MISO_DISP_488* instance as well as the scaling behavior of the three commercial but academically available solvers CPLEX 22.1.1.0 [42], Gurobi 11 [38], and the Cardinal Optimizer COPT 7.2.3 [26]. We solved the instance with PIPS-IPM++ multiple times using a different number of

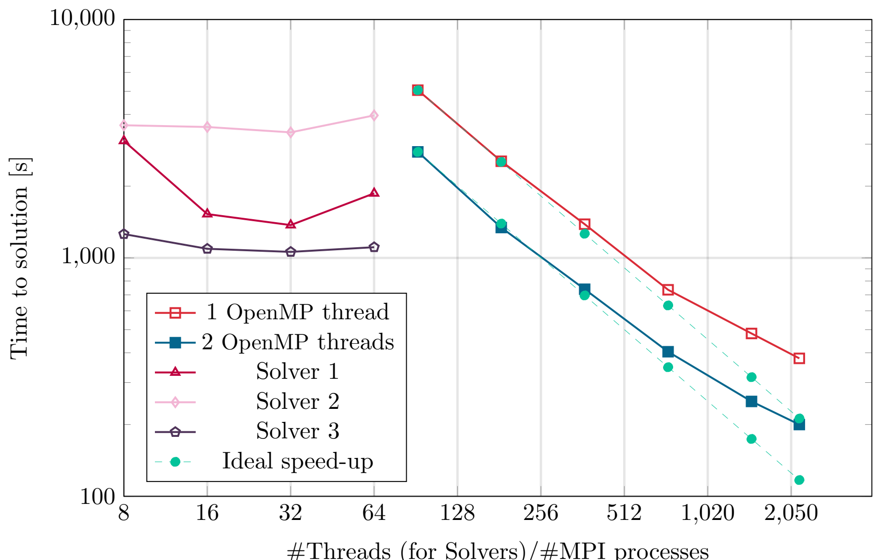


Figure 4: Scaling behavior of all solvers on MISO_DISP_488.

MPI processes and OpenMP threads and (implicitly) cluster nodes. The problem displayed has 120 million nonzeros, 25 million columns, and 12 million rows and is split into 2,190 blocks, which limits the number of usable MPI processes to 2,190. Notably, the instance contains roughly 55,000 linking constraints, and the initial symbolic analysis of the Schur complement in PIPS-IPM++ determined an upper bound of 100,424,258 nonzeros. This made the instance intractable for the original PIPS-IPM++ algorithm (see table 1). We adapted the number of threads for each solution with the commercial solvers while turning off crossover and using barrier as the optimization algorithm. Both PIPS-IPM++ and the commercial solvers were set to solve the LP up to a convergence tolerance of 10^{-6}.

Additionally, we plot the optimal speed-up for each scenario as a dotted line. Our approach is not expected to achieve linear speed-up, as the mid-level Schur complements are solved by only a subset of threads in parallel, and the root-level Schur complement is solved sequentially. However, HSCA achieves a strong near-linear speed-up on the given instance. Additionally, for this instance PIPS-IPM++ benefits from adding a second OpenMP thread while solving. More OpenMP threads did not produce much speed-up over the 2-thread variant and the usefulness of OpenMP is generally problem dependent. It can also be observed, that the commercial solvers do not scale well with the availability of more compute resources. Each solver runs fastest with 32 threads, which we attribute to the memory-boundedness of the Cholesky factorization underlying the IPM.

Impact of block sizes and splitting on the performance of hierarchical approach vs. original approach

While we have previously demonstrated near-ideal scaling on instances suited for parallelization, in this section we deliberately examine instances where the growth of the Schur complement dominates. This highlights the strong-scaling trade-offs and limitations inherent in both the original and hierarchical approaches. We display the different negative impact to be expected from growing Schur complements for PIPS-IPM++ and HSCA.

The performance of both the original PIPS-IPM++ and HSCA is heavily problem-dependent. Given the superior performance of PARDISO on large linear systems, PIPS-IPM++ is largely favored on systems with a manageable number of linking constraints and large blocks of constraints. On the other hand, the hierarchical extension struggles with large Schur complements and large diagonal blocks since MA57's factorization routine is slower on large systems.

Additionally, on a given problem the performance of either approach is decomposition dependent. A "good" decomposition balances two competing effects: coarser blocks reduce the size of the Schur complement but increase diagonal block factorization time, whereas finer blocks decrease diagonal block factorization time but increase both the size of the global Schur complement and communication overhead among MPI processes. For the original approach, the time spent in the Schur complement is highly sensitive to a growing number of linking constraints, whereas for HSCA, this impact is mitigated by the additional Schur complement parallelization. We note that for neither approach does using the finest possible splitting yield the best runtime. While finer splitting reduces the size of diagonal blocks, it enlarges the global Schur complement and increases communication overhead among the growing number of MPI processes. As a result, runtime can increase with more processors—an expected strong-scaling limitation in Schur complement methods.

In our view, the two approaches complement one another, each being efficient for distinct problem classes. The performance will depend on how a modeler or automatic structure detection algorithm decomposes a given model into blocks: the original approach will favor few large diagonal blocks and a manageable Schur complement; HSCA will favor more and smaller diagonal blocks even if this results in larger Schur complements.

To demonstrate this, we conducted a simple experiment on four small to medium SIMPLE instances with different sizes (fig. 5). These four instances illustrate the behavior under Schur complement stress. From the top left to the bottom right, the instances become increasingly more difficult to solve, simply due to a growing size of their diagonal blocks. The number of linking constraints in the problem, when split into a certain number of blocks, remains constant over all instances. Each instance can theoretically be split into 8,760 blocks. Combining multiple blocks into one decreases the number of linking constraints in the Schur complement approach. We solved each instance with the original PIPS-IPM++ and our new approach, split into five different amounts of blocks: 365, 730, 1,460, 2,920, and 8,760. Here, we always use as many MPI processes as blocks available. In this setting, a lower number of blocks and processes mean bigger diagonal blocks and, as pointed out before, smaller global Schur complements.

First, we see that the overall performance of the original approach for the

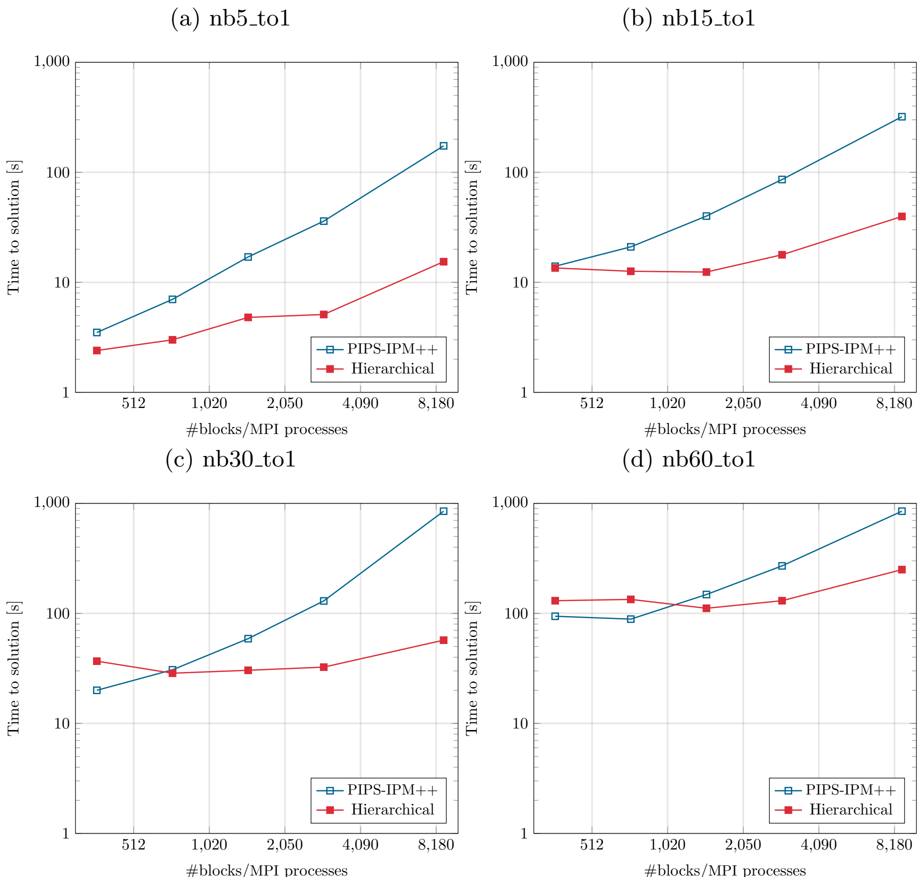


Figure 5: Comparison of the original PIPS-IPM++ and the hierarchical approach for different block sizes and constant amount of linking constraints.

given instances suffers with finer decomposition and only for the largest one can obtain some speedup when more blocks and processes are used. The solution time here is dominated by the time spent in the Schur complement. Only on the largest instance does the speed-up gained by using smaller diagonal blocks outweigh the growing Schur complement cost.

The behavior of HSCA is similar only for the first instance. On the other three instances there is some potential in using more and smaller diagonal blocks, a behavior directly coupled to HSCA's sensitivity to the Schur complement size. In HSCA (in an ideal setting), the number of local linking constraints in each of the Schur complements grows only in $\mathcal{O}(\sqrt{n})$ (with n denoting the number of local linking constraints), as compared to $\mathcal{O}(n)$ for the original approach. HSCA is thus able to scale "further". Still, beyond a certain splitting, all instances experience some slow-down in HSCA.

Comparing the runtime of both algorithms against each other, the original approach benefits from large diagonal blocks but experiences a significant slowdown with a growing number of linking constraints and shrinking diagonal blocks. At the same time, HSCA is worse at handling larger diagonal blocks and large Schur complements. While it can outperform the original PIPS-IPM++ for the smaller two instances, HSCA falls behind as the diagonal block sizes increase. For the hardest (in terms of diagonal block size) two instances, the original PIPS-IPM++ cannot be outperformed by HSCA. In the other two instances, where the ratio of diagonal block size to the number of linking constraints favors HSCA, the original algorithm consistently underperforms. The optimal approach strongly depends on the individual layout of the given instance and the decomposition used.

There are two scenarios where we expect HSCA, given sufficient compute resources, to consistently outperform the original PIPS-IPM++. First, should a user not have access to a high-performance (commercial) sparse direct linear solver like PARDISO but to (say) MA57, HSCA should be picked over the original PIPS-IPM++. A more lightweight solver will benefit from smaller matrices and scale well within HSCA. Second, in distributed but memory-bound environments, such as the JUWELS supercomputer, large Schur complements can usually not be factorized practically or efficiently, and the only solution method available is the hierarchical extension. In all other circumstances, the out-performance of either approach will be model-dependent.

Solution times of hierarchical approach on intractable models

Lastly, we present experiments on models that the original implementation of PIPS-IPM++ could not solve efficiently. In table 1, we compare solution times obtained with HSCA, the original PIPS-IPM++, and the three commercial solvers, CPLEX 22.1.1.0, Gurobi 11, and COPT 7.2.3 each run with 32 threads. Apart from the run time, we list model properties for each instance: the number of variables (*vars.*), constraints (*cons.*), nonzeros (*nnz.*), and linking constraints (*link. cons.*). Variables, constraints, and nonzeros are given in millions (M). We write MEM whenever PIPS-IPM++ crashes due to memory limitations on the compute nodes. For MISO_DISP_120 and MISO_DISP_488, we could not obtain results with the original PIPS-IPM++ due to errors in underlying software packages. For these same instances, Solver 2 could not obtain optimal solu-

tions but instead returned suboptimal ones. The respective times are marked with '*'. The number of nodes used to obtain these results varies between 16 and 92; the column "procs./nodes" denotes for every instance the number of MPI processes and compute nodes used. For SIMPLE_4, MISO_DISP_120, and MISO_DISP_488, we use more nodes than indicated by the number of MPI processes ($\lceil \frac{\#\text{procs}}{48} \rceil$, 48 cores per node), as we additionally run with two OpenMP threads per process. For SIMPLE_4, HSCA splits the Schur complement into 4 layers and 3 for the other instances.

	Size					Run time (seconds)				
Instance	vars.	cons.	nnz.	link. cons.	procs./nodes	HSCA	PIPS	Solver 1	Solver 2	Solver 3
SIMPLE_1	5.79M	5.26M	20.94M	525 600	730/16	561	849	461	704	717
SIMPLE_2	5.78M	5.26M	20.94M	175 200	730/16	121	278	479	710	744
SIMPLE_3	59.80M	51.60M	205.57M	71 680	1 024/43	922	400	4 571	8 076	5 368
SIMPLE_4	76.18M	59.80M	254.72M	1 679 371	2 190/48	2 343	MEM	6 555	7 902	6 555
MISO_EXP_30	4.80M	5.06M	19.42M	96 388	2 190/48	76	MEM	795	855	647
MISO_EXP_120	14.56M	15.35M	57.90M	311 040	2 190/48	741	MEM	4 690	12 391	3 235
MISO_EXP_240	22.37M	26.40M	98.39M	1 156 398	1 472/62	1 392	MEM	12 252	25 391	10 624
MISO_DISP_120	11.32M	12.11M	45.99M	48 240	2 190/92	48	-	627	1 037*	397
MISO_DISP_488	25.44M	11.79M	120.09M	54 844	2 190/92	206	-	1 373	3 363*	1 060

Table 1: Computational results for large-scale instances (M=million).

In line with the results presented in [61], PIPS-IPM++ and HSCA often outperform commercial software. This is especially visible in real-world MISO instances and larger SIMPLE instances. Overall, HSCA can achieve speed-ups of up to a factor of 10 on the presented instances. Similar to fig. 5, we see that for the instances SIMPLE_2 and SIMPLE_3, either approach can outperform the other. This depends on the amount of linking constraints vs. the size of the diagonal blocks. PIPS-IPM++ cannot efficiently solve many instances as it often runs out of memory when factorizing the Schur complement. None of the instances could be solved using the original PIPS-IPM++ and MA57 instead of PARDISO.

We finally note that the performance of the commercial solvers often also strongly depends on their ability to presolve the instances. They usually can reduce the problem size significantly more than PIPS-IPM++. This has mainly two reasons. First, PIPS-IPM++ presolve is designed not to destroy the underlying problem structure of an instance [30]. This mostly leads to limitations when aggregating variables in between blocks. Second, PIPS-IPM++ current presolve only implements a handful of methods–far from what is available in modern commercial LP solvers. As a result, the presolved problems in PIPS-IPM++ are often more challenging and exhibit higher redundancy than their commercial counterparts.

6 Conclusions

This paper presents HSCA, a novel distributed approach for efficiently solving large arrowhead structured LPs. It builds on an efficient distributed factorization of the KKT systems arising in IPMs. The approach exploits the primal–dual block-angular structure of our models to distribute and parallelize the underlying linear algebra. Extending the original PIPS-IPM++ implementation, we applied a multi-level Schur complement decomposition to deal with

problems with up to 1.7 million linking constraints. We showed the limits of our approach and discussed scenarios in which either implementation outperforms the other. While HSCA often outperforms commercial solvers and the original PIPS-IPM++, its performance is heavily problem-dependent. In this respect, HSCA complements the original PIPS-IPM++ and should be considered an alternative when solving large-scale arrowhead LPs. Good scalability is the key feature that gives PIPS-IPM++ and its hierarchical extension an edge over other commercial optimization software. As demonstrated, most commercial software packages do not scale well with additional threads and cores. As parallel hardware becomes increasingly affordable and efficient, scalable numerical algorithms are likely to outperform and eventually replace traditional solution methods, even when their total computational cost is higher.

While the first implementation of the hierarchical approach has already achieved promising results, several other research paths remain. First, presolve is still limited, and a full presolve suite would improve both run time and stability of our solver and would make it more comparable to commercial software. Second, we are exploring alternative linear solvers that may further improve PIPS-IPM++ performance. Third, as HSCA is not limited to two-link constraints, a more elaborate structure detection could be further implemented. In particular, one could account for linking variables and constraints linking n consecutive blocks, leading to a more complicated Schur complement structure with additional "global" linking constraints in the subsystems. The IPM algorithm itself, discussed only briefly in this paper, also offers room for further improvement. To this end, we plan on implementing a sequential version of PIPS-IPM++ to robustify the IPM and compare its performance against the performance of other codes. Another promising direction is extending HSCA to support approximate preconditioners instead of exact factorizations at each IPM step. Many of the preconditioners mentioned in [4, 21, 45] can directly profit from exploitation of the arrowhead structure and even be extended to a recursive arrowhead structure.

Last, many open questions remain to be addressed on the application side. In the ongoing research project Peregrine[5], we are integrating automatic structure detection, similar to the work done in [7], into our software, as the modeler currently has to annotate a model's block structure manually. Lastly, we are preparing an open model library containing large-scale ESMs from different contexts as well as distribution planning problems, which will also include the problems used in this paper. We invite others to try out our software and contribute; the version used in this paper is available on GitHub.

Acknowledgements

The described research activities are funded by the Federal Ministry for Economic Affairs and Energy within the project UNSEEN (ID: 03EI1004C, 03EI1004D). 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). We thank Charlie Vanaret for fruitful discussions and his fierce effort to make this paper more readable.

[5]https://www.dlr.de/en/ve/research-and-transfer/projects/project-peregrine

References

[1] Alappat et al. A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication. *ACM Trans. Parallel Comput.*, 7(3), June 2020.

[2] Amestoy et al. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM J. Matrix Anal. Appl.*, 23(1):15–41, 2001.

[3] Knud D. Andersen. A modified Schur-complement method for handling dense columns in interior-point methods for linear programming. *ACM Trans. Math. Software*, 22(3):348–356, sep 1996.

[4] Michele Benzi, Gene H. Golub, and Jörg Liesen. Numerical solution of saddle point problems. *Acta Numer.*, 14:1–137, 2005.

[5] Timo Berthold. *Primal Heuristics for Mixed Integer Programs.* PhD thesis, Technical University Berlin, 01 2006.

[6] Robert E. Bixby. Solving Real-World Linear Programs: A Decade and More of Progress. *Oper. Res.*, 50(1):3–15, 2002.

[7] Ralf Borndörfer, Carlos E. Ferreira, and Alexander Martin. Decomposing Matrices into Blocks. *SIAM J. Optim.*, 9(1):236–269, 1998.

[8] K. Cao, J. Metzdorf, and S. Birbalta. Incorporating Power Transmission Bottlenecks into Aggregated Energy System Models. *Sustainability*, 10(6):1–32, 2018.

[9] Jordi Castro. A Specialized Interior-Point Algorithm for Multicommodity Network Flows. *SIAM J. Optim.*, 10(3):852–877, 2000.

[10] Jordi Castro. Interior-point solver for convex separable block-angular problems. *Optim. Methods Softw.*, 31(1):88–109, 2016.

[11] Jordi Castro, Laureano F. Escudero, and Juan F. Monge. On solving large-scale multistage stochastic optimization problems with a new specialized interior-point approach. *European J. Oper. Res*, 310(1):268–285, 2023.

[12] Ümit Çatalyürek and Cevdet Aykanat. *PaToH (Partitioning Tool for Hypergraphs)*, pages 1479–1487. Springer US, Boston, MA, 2011.

[13] V. Chvátal. *Linear Programming.* Series of books in the mathematical sciences. W. H. Freeman, 1983.

[14] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI Message Passing Interface Standard. In *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218, 1994.

[15] Marco Colombo and Jacek Gondzio. Further development of multiple centrality correctors for interior point methods. *Comput. Optim. Appl.*, 41(3):277–305, Dec 2008.

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

[17] Leonardo Dagum and Ramesh Menon. OpenMP: an industry standard API for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, 1998.

[18] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Univ. Press, 01 2017.

[19] Iain S. Duff. MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Software*, 30(2):118–144, jun 2004.

[20] Iain S. Duff and John Ker Reid. MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations, 1982.

[21] Marco D'Apuzzo, Valentina De Simone, and Daniela di Serafino. On mutual impact of numerical linear algebra and large-scale optimization with focus on interior point methods. *Comput. Optim. Appl.*, 45(2):283–310, December 2008.

[22] Schlag et al. High-Quality Hypergraph Partitioning. *ACM J. Exp. Algorithmics*, 27, February 2023.

[23] Robert Fourer. Solving staircase linear programs by the simplex method, 1: Inversion. *Math. Program.*, 23(1):274–313, December 1982.

[24] Friedlander et al. Optimization with staircase structure: An application to generation scheduling. *Comput. Oper. Res.*, 17(2):143–152, 1990.

[25] Gerald Gamrath and Marco E. Lübbecke. *Experiments with a Generic Dantzig-Wolfe Decomposition for Integer Programs*, page 239–252. Springer Berlin Heidelberg, 2010.

[26] Ge et al. Cardinal Optimizer (COPT) User Guide, 2023.

[27] Geoffrey et al. A Hierarchical Low Rank Schur Complement Preconditioner for Indefinite Linear Systems. *SIAM J. Sci. Comput.*, 40(4):A2234–A2252, 2018.

[28] E. Michael Gertz and Stephen J. Wright. Object-Oriented Software for Quadratic Programming. *ACM Trans. Math. Software*, 29(1):58–81, March 2003.

[29] Gils et al. Integrated modelling of variable renewable energy-based power supply in Europe. *Energy*, 123:173–188, 2017.

[30] Gleixner et al. First Experiments with Structure-Aware Presolving for a Parallel Interior-Point Method. In *Oper. Res. Proc.*, pages 105–111. Springer, 2020.

[31] Jacek Gondzio. Interior point methods 25 years later. *European J. Oper. Res.*, 218(3):587–601, 2012.

[32] Jacek Gondzio. Interior point methods in the year 2025. *EURO J. Comput. Optim.*, 13:100105, 2025.

[33] Jacek Gondzio and Andreas Grothey. Solving Distribution Planning Problems with the Interior Point Method. Technical report, The University of Edinburgh, 02 2006.

[34] Jacek Gondzio and Andreas Grothey. Solving non-linear portfolio optimization problems with the primal-dual interior point method. *European J. Oper. Res*, 181(3):1019–1029, 2007.

[35] Jacek Gondzio and Andreas Grothey. Exploiting structure in parallel implementation of interior point methods for optimization. *Comput. Manag. Sci.*, page 135–160, 2009.

[36] Michael D. Grigoriadis and Leonid G. Khachiyan. An Interior Point Method for Bordered Block-Diagonal Linear Programs. *SIAM J. Optim.*, 6(4):913–932, 1996.

[37] Anshul Gupta. WSMP: Watson Sparse Matrix Package. Part I - Direct Solution of Symmetric Sparse Systems Version 1.0.0, 2000.

[38] Gurobi Optimization, LLC. Gurobi Optimizer, 2023.

[39] Leonard Göke. A graph-based formulation for modeling macro-energy systems. *Applied Energy*, 301:117377, 2021.

[40] Hoersch et al. PyPSA-Eur: An open optimisation model of the European transmission system. *Energy Strategy Reviews*, 22:207–215, 2018.

[41] J. Hogg and Jennifer A. Scott. An indefinite sparse direct solver for large problems on multicore machines. *Rutherford Appleton Laboratory Technical Reports*, 2010.

[42] IBM ILOG. CPLEX 22.1.2, 2024.

[43] Intel Corporation. Intel Math Kernel Library, 2024.

[44] Jülich Supercomputing Centre. JUWELS: Modular Tier-0/1 Supercomputer at the Jülich Supercomputing Centre. *Journal of large-scale research facilities*, 5(A135), 2019.

[45] Samah Karim and Edgar Solomonik. Efficient Preconditioners for Interior Point Methods via a New Schur Complement-Based Strategy. *SIAM J. Matrix Anal. Appl.*, 43(4):1680–1711, 2022.

[46] Karypis et al. Multilevel hypergraph partitioning: application in VLSI domain. In *Proc. 34th Annual Design Automat. Conf*, DAC '97, page 526–529. ACM Press, 1997.

[47] Kibaek Kim, Cosmin G. Petra, and Victor M. Zavala. An Asynchronous Bundle-Trust-Region Method for Dual Decomposition of Stochastic Mixed-Integer Programming. *SIAM J. Optim.*, 29(1):318–342, 2019.

[48] Lubin et al. Scalable stochastic optimization of complex energy systems. In *11 ACM/IEEE Supercomputing Conference*, 2011.

[49] Lubin et al. On parallelizing dual decomposition in stochastic integer programming. *Oper. Res. Lett.*, 41(3):252–258, 2013.

[50] Lueg et al. Domain decomposition preconditioners for schur complement systems arising in structured nonlinear optimization problems. *Optim. Eng.*, September 2025.

[51] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. On Implementing Mehrotra's Predictor–Corrector Interior-Point Method for Linear Programming. *SIAM J. Optim.*, 2(3):435–449, 1992.

[52] István Maros and Csaba Mészáros. The role of the augmented system in interior point methods. *European J. Oper. Res*, 107(3):720–736, 1998.

[53] Tine Meersman, Broos Maenhout, and Koen Van Herck. A nested Benders decomposition-based algorithm to solve the three-stage stochastic optimisation problem modeling population-based breast cancer screening. *European J. Oper. Res*, 310(3):1273–1293, 2023.

[54] Sanjay Mehrotra. On the Implementation of a Primal-Dual Interior Point Method. *SIAM J. Optim.*, 2(4):575–601, 1992.

[55] Csaba Mészáros. Detecting "dense" columns in interior point methods for linear programs. *Comput. Optim. Appl.*, 36(2–3):309–320, apr 2007.

[56] Pacaud et al. Parallel interior-point solver for block-structured nonlinear programs on SIMD/GPU architectures. *Optim. Methods Softw.*, 39(4):874–897, 2024.

[57] Petra et al. An Augmented Incomplete Factorization Approach for Computing the Schur Complement in Stochastic Optimization. *SIAM J. Sci. Comput.*, 36(2):C139–C162, 2014.

[58] Florian A. Potra and Stephen J. Wright. Interior-point methods. *J. Comput. Appl. Math.*, 124(1):281–302, 2000.

[59] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of Interior-Point Methods to Model Predictive Control. *J. Optim. Theory Appl.*, 99(3):723–757, December 1998.

[60] Tim Rees and Chen Greif. A Preconditioner for Linear Systems Arising From Interior Point Optimization Methods. *SIAM J. Sci. Comput.*, 29(5):1992–2007, 2007.

[61] Rehfeldt et al. A massively parallel interior-point solver for LPs with generalized arrowhead structure, and applications to energy system models. *European J. Oper. Res.*, 296(1):60–71, 2022.

[62] Y. Saad and B. Suchomel. ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numer. Linear Algebra Appl.*, 9(5):359–378, 2002.

[63] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, January 2003.

[64] Yousef Saad and Jun Zhang. BILUM: Block Versions of Multielimination and Multilevel ILU Preconditioner for General Sparse Linear Systems. *SIAM J. Sci. Comput.*, 20(6):2103–2121, January 1999.

[65] Scholz et al. Speeding up Energy System Models - a Best Practice Guide. Technical report, BEAM-ME project, 06 2020.

[66] Marc C. Steinbach. *Hierarchical Sparsity in Multistage Stochastic Programs*, pages 385–410. Springer US, Boston, MA, 2001.

[67] Tamás Terlaky. *Interior Point Methods of Mathematical Programming*, volume 5. Springer Science & Business Media, 2013.

[68] Manuel Wetzel, Hans Christian Gils, and Valentin Bertsch. Green energy carriers and energy sovereignty in a climate neutral European energy system. *Renewable Energy*, 210:591–603, 2023.

[69] Wetzel et al. REMix: A GAMS-based framework for optimizing energy system models. *Journal of Open Source Software*, 9(99):6330, 2024.

[70] Wiese et al. Balmorel open source energy system model. *Energy Strategy Reviews*, 20:26–34, 2018.

[71] Robert J. Wittrock. *Dual nested decomposition of staircase linear programs*, page 65–86. Springer Berlin Heidelberg, 1985.

[72] Word et al. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Comput. Optim. Appl.*, 59(3):667–688, April 2014.

[73] Stephen J. Wright. *Primal-Dual Interior-Point Methods.* SIAM, 1997.

[74] Stephen J. Wright. Stability of Augmented System Factorizations in Interior-Point Methods. *SIAM J. Matrix Anal. Appl.*, 18(1):191–222, 1997.

[75] Yuanzhe Xi, Ruipeng Li, and Yousef Saad. An Algebraic Multilevel Preconditioner with Low-Rank Corrections for Sparse Symmetric Matrices. *SIAM J. Matrix Anal. Appl.*, 37(1):235–259, 2016.

[76] Xu et al. parGeMSLR: A parallel multilevel Schur complement low-rank preconditioning and solution package for general sparse matrices. *Parallel Comput.*, 113:102956, 2022.

[77] Filippo Zanetti and Jacek Gondzio. A New Stopping Criterion for Krylov Solvers Applied in Interior Point Methods. *SIAM J. Sci. Comput.*, 45(2):A703–A728, 2023.

[78] Victor M. Zavala, Carl D. Laird, and Lorenz T. Biegler. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chem. Eng. Sci.*, 63(19):4834–4845, 2008.

[79] Zhang et al. A stabilised Benders decomposition with adaptive oracles for large-scale stochastic programming with short-term and long-term uncertainty. *Comput. Oper. Res.*, 167:106665, 2024.

A Structure detection within the Schur complement

After realizing that many linking constraints only link two consecutive blocks as described in section 3.3, we can utilize this fact to detect structure in the Schur complement line 3 in algorithm 2 itself. Given the system matrix eq. (10) and defining

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

the Schur complement contributions of each block i have the following structure:

$$\begin{aligned} L_i^T K_i^{-1} L_i &= \begin{bmatrix} 0 & A_i^T \\ 0 & 0 \\ \hat{F}_i & 0 \\ \mathbf{F}_i & 0 \end{bmatrix} \begin{bmatrix} \tilde{K}^i_{1,1} & \tilde{K}^i_{1,2} \\ \tilde{K}^i_{2,1} & \tilde{K}^i_{2,2} \end{bmatrix} \begin{bmatrix} 0 & 0 & \hat{F}_i^T & \mathbf{F}_i^T \\ A_i & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} A_i^T \tilde{K}^i_{2,2} A_i & 0 & A_i^T \tilde{K}^i_{2,1} \hat{F}_i^T & A_i^T \tilde{K}^i_{2,1} \mathbf{F}_i^T \\ 0 & 0 & 0 & 0 \\ \hat{F}_i \tilde{K}^i_{1,2} A_i & 0 & \hat{F}_i \tilde{K}^i_{1,1} \hat{F}_i^T & \hat{F}_i \tilde{K}^i_{1,1} \mathbf{F}_i^T \\ \mathbf{F}_i \tilde{K}^i_{1,2} A_i & 0 & \mathbf{F}_i \tilde{K}^i_{1,1} \hat{F}_i^T & \mathbf{F}_i \tilde{K}^i_{1,1} \mathbf{F}_i^T \end{bmatrix}. \end{aligned}$$

The two-links imply additional structure on $\hat{F}_i \tilde{K}^i_{1,1} \hat{F}_i^T$:

$$\hat{F}_i \tilde{K}^i_{1,1} \hat{F}_i^T = \begin{bmatrix} \vdots \\ 0 \\ \mathcal{F}'_{i-1} \\ \mathcal{F}_i \\ 0 \\ \vdots \end{bmatrix} \tilde{K}^i_{1,1} \begin{bmatrix} \cdots & 0 & \mathcal{F}'^T_{i-1} & \mathcal{F}_i^T & 0 & \cdots \end{bmatrix} =$$

$$\begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \iddots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & \mathcal{F}'_{i-1} \tilde{K}^i_{1,1} \mathcal{F}'^T_{i-1} & \mathcal{F}'_{i-1} \tilde{K}^i_{1,1} \mathcal{F}_i^T & 0 & \cdots \\ \cdots & 0 & \mathcal{F}_i \tilde{K}^i_{1,1} \mathcal{F}'^T_{i-1} & \mathcal{F}_i \tilde{K}^i_{1,1} \mathcal{F}_i^T & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \iddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Summing up all individual Schur complement contributions (and after appropriate symmetric permutation), the final Schur complement is expressed as the following block matrix:

$$P^T \Big(\sum_{i=1}^{N} L_i^T K_i^{-1} L_i \Big) P = \begin{bmatrix} M_1 & H_1^T & & & & E_1^T & 0 \\ H_1 & M_2 & H_2^T & & & E_2^T & 0 \\ & H_2 & \ddots & \ddots & & \vdots & \vdots \\ & & \ddots & & H_{N-2}^T & E_{N-2}^T & 0 \\ & & & H_{N-2} & M_{N-1} & E_{N-1}^T & 0 \\ E_1 & E_2 & \cdots & E_{N-1} & E_{N-1} & M_0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{17}$$

The submatrices in eq. (17) are defined as

$$M_i := \mathcal{F}_i \tilde{K}^i_{1,1} \mathcal{F}_i^T + \mathcal{F}'_i \tilde{K}^{i+1}_{1,1} \mathcal{F}_i'^T,$$

$$E_i := \begin{bmatrix} \mathbf{F}_i \tilde{K}^i_{1,1} \mathcal{F}_i^T + \mathbf{F}_{i+1} \tilde{K}^{i+1}_{1,1} \mathcal{F}_i'^T \\ A_i^T \tilde{K}^i_{2,1} \mathcal{F}_i^T + A_{i+1}^T \tilde{K}^{i+1}_{2,1} \mathcal{F}_i'^T \end{bmatrix}, \tag{18}$$

for $i \in \{1, \dots, N-1\}$ and

$$H_i := \mathcal{F}_{i+1} \tilde{K}^{i+1}_{1,1} \mathcal{F}_i'^T,$$

$$M_0 := \sum_{i=1}^{N} \begin{bmatrix} \mathbf{F}_i \tilde{K}^i_{1,1} \mathbf{F}_i^T & \mathbf{F}_i \tilde{K}^i_{1,2} A_i \\ A_i^T \tilde{K}^i_{2,1} \mathbf{F}_i^T & A_i^T \tilde{K}^i_{2,2} A_i \end{bmatrix}, \tag{19}$$

for $i \in \{1, \dots, N-2\}$. This structure is detected in PIPS-IPM++ and used to precompute the sparsity pattern of the final Schur complement. As long as no additional structure can be detected in the constraints, all blocks in eq. (17) are dense; in particular, eq. (18) and eq. (19) can be significantly large, depending on the number of linking constraints and global linking variables in the problem. We reiterate an upper bound on the number of nonzeros in the Schur complement, as introduced in Observation 1 in [61]:

$$\sum_{i=1}^{N-1} l_i^2 + 2 \sum_{i=1}^{N-2} l_i l_{i+1} + 2 \sum_{i=1}^{N-1} l_i (m_{\mathbf{F}} + n_0) + (m_{\mathbf{F}} + n_0)^2.$$

Even though a two-link structure for linking variables could also be computed, linking variables are seldom local in the models considered. Therefore, this has not been implemented.

The Schur complements of systems without global linking constraints, as is the case for the inner linear systems defined by K^{dense} in eq. (12) and K_i^{inner} in eq. (16) have the band diagonal form

$$\begin{bmatrix} \tilde{M}_1 & \tilde{H}_1^T & & & \\ \tilde{H}_1 & \tilde{M}_2 & \tilde{H}_2^T & & \\ & \tilde{H}_2 & \ddots & \ddots & \\ & & \ddots & & \tilde{H}_{\tilde{N}-2}^T \\ & & & \tilde{H}_{\tilde{N}-2} & \tilde{M}_{\tilde{N}-1} \end{bmatrix}, \tag{20}$$

with $\tilde{N} \in \mathbb{N}$ and $\tilde{M}_i \in \mathbb{R}^{\tilde{l}_i \times \tilde{l}_i}$ and $\tilde{H}_i \in \mathbb{R}^{\tilde{l}_{i+1} \times \tilde{l}_i}$ defined equivalently. These Schur complements are mostly sparse. The number of nonzeros of eq. (20) is thus bounded by $\sum_{i=1}^{\tilde{N}-1} \tilde{l}_i^2 + 2 \sum_{i=1}^{\tilde{N}-2} \tilde{l}_i \tilde{l}_{i+1}$.