Sustainable Energy, Grids and Networks 31 (2022) 100789

Contents lists available at ScienceDirect

Sustainable Energy, Grids and Networks

journal homepage: www.elsevier.com/locate/segan

Massively parallel data analytics for smart grid applications☆

Juraj Kardoš [a,*], Timothy Holt [a], Vincenzo Fazio [b], Luca Fabietti [b], Filippo Spazzini [b], Olaf Schenk [a]

[a] Università della Svizzera italiana, Via Buffi 13, 6900, Lugano, Switzerland
[b] DXT Commodities SA, Via Trevano 2, , 6900, Lugano, Switzerland

ARTICLE INFO

Article history:
Received 3 December 2021
Received in revised form 27 April 2022
Accepted 22 May 2022
Available online 30 May 2022

Keywords:
High-throughput scheduling
Massively parallel computing
Numerical optimization
Power grid
Optimal power flow
Unit commitment

ABSTRACT

Complexity involved in operating modern power and energy systems is constantly increasing given the volatility induced by the rapid integration of intermittent renewable energy sources. In order to operate the power grid in secure and reliable way, a plethora of uncertain parameters need to be considered and hundreds of thousands of different power grid scenarios need to be rapidly evaluated. This works analyzes the computational aspects in massively parallel simulations from the perspective of efficient hardware utilization. A method for efficiently managing and processing the computational tasks is presented, carefully considering the level of parallelism in order to avoid computational bottlenecks and efficiently utilizing modern multicore architectures with deep memory hierarchies. An extensive set of numerical experiments is presented, considering multiple aspects of the computational pipeline. The numerical experiments are performed using mathematical models typically used in the power grid problems, including linear and quadratic programs as well as the models containing the discrete variables. The optimized high-throughput computation strategy has been shown to significantly reduce response times by preventing the memory bottlenecks for various computational models.

1. Introduction

Recent trends in the power grid operations and integration of intermittent renewable energy sources (RES) impose great demand on computational resources. The large number of power grid scenarios that need to be analyzed require not only parallel processing algorithms but also efficient execution strategies for a large number of loosely coupled tasks. These can improve utilization of the computational infrastructure required by individual jobs that need to be scheduled on the available computing resources.

1.1. Motivation and background

In order to operate the power grid in a secure and reliable way, a plethora of parameters need to be considered. These are parameters such as weather, fuel prices, and available transmission capacity between market zones [1]. The difficultly of accurately modeling these systems is constantly increasing given the volatility induced by the rapid integration of intermittent RES into grids. Additional challenges are associated with uncertainty quantification of the model parameters and their sensitivity analysis. All these factors contribute to a large number of power grid scenarios that need to be rapidly solved in order to provide power grid operators tools required to control the complex power grid systems and manage the associated uncertainty.

Most of these problems can benefit from parallel processing, often built into the simulation frameworks or available in the off the shelf solvers such as HiOP [2], PIPS [3], or Beltistos [4]. Nonetheless, the question of the optimal level of parallelism arises and is left up to the end user to decide. This decision, however, requires the knowledge of the underlying architecture, since the excessive level of parallelism might introduce many bottlenecks on the hardware level and result in a significant slowdown of the overall processing time. This work provides an analysis of the computational setup that could guide users of such parallel tools and help to achieve high-throughput data analytics.

1.2. Research context

Power grid dynamics are typically modeled as multistage stochastic unit commitment (UC) problems [5–7], optimal power flow (OPF) [8,9] problems, or economic dispatch (ED) [10,11].

☆ This research is part of the activities of the Innosuisse project no 34394.1 entitled "High-Performance Data Analytics Framework for Power Markets Simulation" and the Swiss Centre for Competence in Energy Research on the Future Swiss Electrical Infrastructure (SCCER-FURIES), which is financially supported by the Swiss Innovation Agency (Innosuisse - SCCER program).

* Corresponding author.
E-mail address: juraj.kardos@usi.com (J. Kardoš).

https://doi.org/10.1016/j.segan.2022.100789

Nomenclature

C^{S}, C^{G}	Storage device and generator connectivity matrices
$\boldsymbol{G}$	Set of generation units
$\boldsymbol{\theta}$	Bus voltage angles
$\mathbf{p}^{\mathrm{D}}$	Active power demands
$\mathbf{p}^{\mathrm{G}}$	Active power injections, including conventional generators and storages
$\mathbf{p}^{\mathrm{gen}}$	Active power injections from conventional generators
$\mathbf{p}^{\mathrm{G}}_{\mathrm{min}}$, $\mathbf{p}^{\mathrm{G}}_{\mathrm{max}}$	Active generation box bounds
$\mathbf{p}^{\mathrm{G,on}}$, $\mathbf{p}^{\mathrm{G,off}}$	Ramp up and down of conventional generators
$\boldsymbol{\Delta}^{\mathrm{up}}$, $\boldsymbol{\Delta}^{\mathrm{lo}}$	Ramp up and down limits
$\mathbf{p}^{\mathrm{S}}$	Composite vector of the storage injections including discharge and charge powers
$\mathbf{p}^{\mathrm{Sd}}$, $\mathbf{p}^{\mathrm{Sc}}$	Storage discharge and charge power injections
$\mathbf{p}^{\mathrm{S}}_{\mathrm{min}}$, $\mathbf{p}^{\mathrm{S}}_{\mathrm{max}}$	Minimum and maximum active storage power output
$\mathbf{n}^{v}$	Zone power flows — import/export
$\mathbf{p}^{\mathrm{B}}$	Bus power flows
$\mathbf{s}^{\mathrm{max}}_{\mathrm{L}}$	Branch power flow limits
$\boldsymbol{c}^{f}$	Cost coefficients in LP
$f(\cdot)$	Objective function in QP and MIP
$\boldsymbol{c}_{\varepsilon}(\cdot)$	DC power flow balance constraints
$\boldsymbol{c}_{l}(\cdot)$	Line flow constraints
$\boldsymbol{u}$	On-line status of the generators
$\mathbf{v}$, $\boldsymbol{w}$	Startup and shutdown states
τ^{u}, τ^{d}	Minimum up/down times of generator units
$\boldsymbol{\epsilon}^{\mathrm{max}}_{\mathrm{S}}$	Capacity of the storage devices
$\boldsymbol{\epsilon}_{0}$	Initial state of charge of the storage devices
$\boldsymbol{\epsilon}$	State of charge of the storage devices
η_{d}, η_{c}	Discharging and charging efficiency
$\mathbf{B}^{\mathrm{S}}$	Storage efficiency matrix
$\boldsymbol{T}$, N	Set of time periods and its cardinality
N_{S}	Number of storage devices
δt	Time period length
n	Number of overall jobs to process
c	Number of concurrent workers
t^{c}_{avg}	Average time of processing a single job when running c workers simultaneously
t^{*}_{n}	Lower bound on processing time of n jobs
t_{n}	Expected processing time of n jobs
ψ	Slowdown of t^{c}_{avg} relative to single worker execution t^{1}
ϕ	Smoothed rate of change of ψ
$[\cdot]$	Operator representing concatenation of column vectors $\left[x_1, x_2, \ldots, x_n\right] = \left[x_1^{\mathrm{T}}, x_2^{\mathrm{T}}, \ldots, x_n^{\mathrm{T}}\right]^{\mathrm{T}}$

Uncertainty is incorporated using stochastic programming techniques based on scenario trees in which the uncertainty is known at each node. After applying various scenario generation schemes based either on expert knowledge, artificial intelligence, or Monte Carlo simulation, the stochastic UC becomes a large optimization problem. Due to the large-scale nature of the problem, the computational complexity is addressed by decomposing the problem into smaller subproblems and utilizing parallel processing. Benders decomposition, Lagrangian relaxation, augmented Lagrangian methods, or progressive hedging are usually among the methods of choice [12,13], since parallelization of these solution algorithms is straightforward.

Stochastic UC is a risk-neutral model that is concerned with the optimization of expected payoff. In order to enable risk modeling, individual stochastic trajectories, represented by a single UC problem, are analyzed independently. Similarly as before, a large number of model evaluations are required. With this method one can obtain all possible price trajectories resulting from the selected scenarios. The resulting price distributions inform the risk management processes that are crucial to applications such as energy trading [14].

Additionally, many model parameters are determined experimentally, using the expert knowledge or based on historical data, with the exact values not available. In order to properly evaluate the effects of the uncertain parameters, one needs to perform an uncertainty quantification [15–17]. It quantifies the confidence of the model output given the uncertainty in the model parameters. Sensitivity analysis is tightly linked to uncertainty quantification and is the process of quantifying the fraction of the output uncertainty that can be explained by individual parameters. However, global sensitivity analysis presents computational challenges due to the large number of input–output samples needed to estimate the uncertainty contributions. In order to perform the sensitivity analysis, parallel and high-throughput processing techniques are essential.

Similarly to the continuous counterparts, the mixed integer linear programming (MIP) models are used in the stochastic frameworks. Many aspects of real-life problems are modeled using discrete variables, including on-line status of generator units [6,7], transformer tap ratios [18], models of the storage devices [7], or demand flexibility models [19,20]. The resulting MIP problems are solved using algorithms usually based on the dual simplex (DS) or interior point (IP) method extended by heuristics to deal with the integrality constraints, including branch and cut, cutting planes, or many advanced presolving techniques. As such, these algorithms have different memory access patterns and might behave differently compared to the underlying algorithm for the continuous variables.

High-throughput processing is usually supported by an appropriate software tool, either classical job schedulers such as SLURM [21] or the workload meta schedulers. These include FireWorks [22], supporting also dynamic workflows, failure-detection routines, and built-in tools and execution modes for running high-throughput computations at large computing centers. Launcher [23] is a utility for performing simple, data parallel, high-throughput computing workflows on clusters, massively parallel processor systems, workgroups of computers, and personal machines. GREASY [24] meta scheduler is used to manage high-throughput simulations and to simplify the execution of embarrassingly parallel simulations in any environment. It was primarily designed to run serial applications. GREASY is used in this work due to its architectural simplicity, ease of use, and the fact that it is a tool already used at multiple supercomputing centers.

1.3. Relevant literature

Massively parallel simulations that exploit modern multi-core central processing units (CPUs) introduce pressure on various

subsystems, particularly main memory. Existing literature on efficient utilization of the available computational resources and avoidance of the bottlenecks is very sparse and scattered across various scientific domains. Sources of bottlenecks for parallel protocol processing and high-speed data transfers are identified, e.g., in [25,26]. The work in [25] studies the impact of different process affinity strategies, considering affinity using cores within the same or different sockets. The authors conclude that affinitization has a significant impact on parallel protocol processing efficiency, and that the performance bottleneck changes significantly with different affinitization strategies. The focus is put on the communication patterns as opposed to computationally heavy tasks which do not require any interprocess communication, which is the main focus of this work.

The study [26] quantifies cache memory limitations on nonuniform memory access multicore platforms arising during parallel optimization of simulation models governed by partial differential equation (PDE). Many parallel tasks are generated within the PDE model calibration problems which seek to find the model parameters that minimize the error between the PDE model and observed reality. Typically, many simulations are run in parallel, each on its own core. Affinity scheduling strategy for parallel computation is proposed, such that the computational efficiency improves due to improved utilization of the memory hierarchy. It is acknowledged that utilizing excessive parallelism does deteriorate the cache utilization, especially when the processes can migrate across the cores. However, the study does not provide any discussion on how to determine the level of parallelism minimizing the memory bottlenecks on top of enforcing the CPU affinity. Additionally, it does not consider the scheduling of the individual jobs on large computational clusters.

1.4. Contributions and organization

This work analyzes high-throughput scheduling techniques and addresses the computational challenges of the massive parallelism associated with stochastic models, uncertainty analysis, or similar applications that rely on a large number of computational tasks which commonly arise in smart grid operations. The main contributions of this paper can be summarized as follows:

- Introduce a technique that mitigates the bottlenecks of embarrassingly parallel simulations by maximizing the utilization of available computational resources and thus reducing the processing time.
- Investigate the proposed technique on various mathematical programs including market based ED, OPF, and UC and experimentally validate the predictions.
- Perform the benchmarks using the ED models of the continental Europe and OPF models of the Swiss transmission network.

This work is based on a previous study of LP problems [14], extending the proposed concepts to additional problem types typically encountered in power grid analysis.

The rest of this paper is organized as follows. Power grid models are introduced in Section 2. The solution, based on both DS and IP methods, is briefly sketched and analyzed from the perspective of computational resource requirements in Section 3. A procedure for determining the optimal level of parallelism is proposed in Section 4. Finally, Section 5 presents numerical experiments analyzing individual stages of the solution algorithms, including computations and data analysis, at a large and distributed scale.

2. Power grid models

Energy markets and operations of the power grid devices are modeled on different levels of abstraction, capturing different aspects of the underlying physical equipment. Some of the most commonly used mathematical models are (i) an ED problem modeled as LP, (ii) direct current (DC) OPF formulated as QP, and (iii) the UC considering also the discrete aspects of the problem modeled as MIP.

2.1. Zone-based market model

The European electricity market is based on bidding zones, which are modeled as one node. Lossless ED considers a problem where the objective is to find the set of generator dispatch points $\mathbf{p}^{\mathrm{G}}$ that minimize the total cost of meeting a specified demand $\mathbf{p}^{\mathrm{D}}$, without modeling any network infrastructure. The problem consists of several zones, where each zone contains several generators $\boldsymbol{G}$ and energy that can be imported or exported. Similarly, the problem is defined over a multiperiod time horizon $\boldsymbol{T}$. The LP model in this work represents a simplified UC problem formulated as a continuous problem in order to have certain guarantees about the convergence and optimality of the solution, as well as reducing the computation time. The LP model reads

$$\underset{\mathbf{p}^{\mathrm{G}}}{\text{minimize}} \quad \sum_{t\in\boldsymbol{T}}\sum_{g\in\boldsymbol{G}} \boldsymbol{c}^{f}_{g,t}\mathbf{p}^{\mathrm{G}}_{g,t} \tag{1a}$$

$$\text{subject to } \forall t \in \{1,2,\ldots,N\}:$$

$$\mathbf{p}^{\mathrm{gen}}_{g,t} = \mathbf{p}^{\mathrm{gen}}_{g,t-1} + \mathbf{p}^{\mathrm{G,on}}_{g,t} - \mathbf{p}^{\mathrm{G,off}}_{g,t}, \tag{1b}$$

$$\sum_{g\in\boldsymbol{G}} \mathbf{p}^{\mathrm{gen}}{}_{g,t} + \mathbf{n}^{v}_{t} + \mathbf{p}^{\mathrm{Sd}}_{t} = \mathbf{p}^{\mathrm{D}}_{t} + \mathbf{p}^{\mathrm{Sc}}_{t}, \tag{1c}$$

$$\mathbf{p}^{\mathrm{G}}_{\min} \le \mathbf{p}^{\mathrm{G}}_{t} \le \mathbf{p}^{\mathrm{G}}_{\max}, \tag{1d}$$

$$\boldsymbol{\epsilon}^{\min}_{\mathrm{S}} \le \boldsymbol{\epsilon}_{t} \le \boldsymbol{\epsilon}^{\max}_{\mathrm{S}}, \tag{1e}$$

$$\mathbf{p}^{\mathrm{G,off}}_{g,t} \le \boldsymbol{\Delta}^{\mathrm{lo}}, \quad \mathbf{p}^{\mathrm{G,on}}_{g,t} \le \boldsymbol{\Delta}^{\mathrm{up}}. \tag{1f}$$

The objective (1a) is to minimize the energy cost, where the cost coefficients $\boldsymbol{c}^{f}$ represent marginal cost and approximate start-up costs of each conventional generation unit $g \in \boldsymbol{G}$. The power injection variables $\mathbf{p}^{\mathrm{G}} = \left[\mathbf{p}^{\mathrm{gen}}, \mathbf{p}^{\mathrm{S}}\right]$ consists not only of the conventional generator outputs $\mathbf{p}^{\mathrm{gen}}$ but also includes the injections $\mathbf{p}^{\mathrm{S}}$ incurred by the storage devices. The conventional power output is represented recursively with respect to the previous time instance and the power increment $\mathbf{p}^{\mathrm{G,on}}_{g,t}$ and decrement $\mathbf{p}^{\mathrm{G,off}}_{g,t}$ in the current time period, as expressed in (1b). Considering this representation, minimum up-/down-time and generation ramp constraints can be easily approximated by additional linear constraints. The demand balance constraint (1c) states that the sum of all generation components (power plants, net import, and storage discharge) should be equal to the sum of all load components (demand and storage charging) for all time instances $t \in \boldsymbol{T}$. The net import $\mathbf{n}^{v}_{t}$ is simply a sum of the power imports and exports for the given zone of interest. Additional constraints are imposed for the links between the zones, such as maximum capacity or flow-based constraints [27].

Energy storage devices are modeled using charging and discharging efficiencies and technical limitations of the state of charge, similar to the model in [28–30]. N_{S} energy storage units are considered, where the vector of the storage power injections consists of discharging and charging injections,

$$\mathbf{p}^{\mathrm{S}} = \left[\mathbf{p}^{\mathrm{Sd}}_{1}, \ldots, \mathbf{p}^{\mathrm{Sd}}_{N_{\mathrm{S}}}, \mathbf{p}^{\mathrm{Sc}}_{1}, \ldots, \mathbf{p}^{\mathrm{Sc}}_{N_{\mathrm{S}}}\right]. \tag{2}$$

The evolution of the state of charge levels $\boldsymbol{\epsilon}_{t} \in \mathbb{R}^{N_{\mathrm{S}}}$ follows the update equation

$$\boldsymbol{\epsilon}_{t} = \boldsymbol{\epsilon}_{t-1} + \mathbf{B}^{\mathrm{S}}\,\mathbf{p}^{\mathrm{S},t} \quad t = 1,\ldots,N, \tag{3}$$

and introduces a coupling between the individual time periods. The energy level in each period needs to honor the storage capacity, as expressed by the constraint (1e). The initial storage level is denoted $\boldsymbol{\epsilon}_0$ and the constant matrix $\mathbf{B}^{\mathrm{S}} \in \mathbb{R}^{N_{\mathrm{S}} \times 2N_{\mathrm{S}}}$ models discharging and charging efficiencies of the storage devices,

$$\mathbf{B}^{\mathrm{S}} = -\delta t \begin{pmatrix} \eta_{\mathrm{d},1}^{-1} & & & \eta_{\mathrm{c},1} & & \\ & \ddots & & & \ddots & \\ & & \eta_{\mathrm{d},N_{\mathrm{S}}}^{-1} & & & \eta_{\mathrm{c},N_{\mathrm{S}}} \end{pmatrix} \tag{4}$$

with the discharging and charging efficiencies $\eta_{\mathrm{d},i}$ and $\eta_{\mathrm{c},i}$, $i = 1, 2, \ldots, N_{\mathrm{S}}$.

2.2. Optimal power flow model

An extension of the ED, considering also the transmission network and DC power flow equations $\boldsymbol{c}_{\varepsilon}$ as a function of bus voltage angle variables $\boldsymbol{\theta}$, along with limits on the branch power flows $\boldsymbol{c}_{l}$, becomes the DC OPF problem. The DC OPF is formulated as

$$\underset{\boldsymbol{\theta},\ \mathbf{p}^{\mathrm{G}}}{\text{minimize}} \sum_{t \in \boldsymbol{T}} \sum_{g \in \boldsymbol{G}} f_g(\mathbf{p}_{g,t}^{G}) \tag{5a}$$

subject to $\forall t \in \{1, 2, \ldots, N\}$:

$$\mathbf{c}_{\varepsilon}^{t}(\boldsymbol{\theta}_t, \mathbf{p}_{g,t}^{\mathrm{G}}) = \mathbf{0}, \tag{5b}$$

$$\mathbf{c}_{l}^{t}(\boldsymbol{\theta}_t) \leq \mathbf{p}_{\mathrm{L}}^{\mathrm{S}\max}, \tag{5c}$$

$$\mathbf{p}_{\min}^{\mathrm{G}} \leq \mathbf{p}_{g,t}^{\mathrm{G}} \leq \mathbf{p}_{\max}^{\mathrm{G}}, \tag{5d}$$

$$\boldsymbol{\epsilon}_{\mathrm{S}}^{\min} \leq \boldsymbol{\epsilon}_t \leq \boldsymbol{\epsilon}_{\mathrm{S}}^{\max}, \tag{5e}$$

$$-\boldsymbol{\Delta}^{\mathrm{lo}} \leq \mathbf{p}_{g,t}^{\mathrm{G}} - \mathbf{p}_{g,t-1}^{\mathrm{G}} \leq \boldsymbol{\Delta}^{\mathrm{up}}. \tag{5f}$$

The objective function f_g is a quadratic cost defined for each generation unit $g \in \boldsymbol{G}$. Other cost components might also include the wear and tear of load-following ramping and value of the initial and expected leftover stored energy in the storage devices. At each network bus, the external power injections must equal the injections from the connected generators, storages, and load components, resulting in the power balance constraint (5b)

$$\mathbf{c}_{\varepsilon}^{t} := C^{\mathrm{G}} \mathbf{p}_t^{\mathrm{gen}} + C^{\mathrm{S}} \mathbf{p}_t^{\mathrm{S}} - \mathbf{p}_t^{\mathrm{D}} - \mathbf{p}_t^{\mathrm{B}}(\boldsymbol{\theta}_t), \tag{6}$$

where C^{G}, C^{S} are the generator and storage connectivity matrices, respectively. The power flow in the transmission lines is limited, as expressed by the constraint (5c). The intertemporal coupling is introduced by energy storage devices (5e) and generator ramp limits (5f). Additional modeling aspects are described in more detail in [30–32].

2.3. Unit commitment

The LP and QP problems in the previous sections have been restricted to continuous optimization variables. The real-life decision problems also consist of discrete UC decisions, modeled by integral variables. The problems include additional startup and shutdown costs associated with changes in on-line status from a prior commitment state. In multiperiod problems, these states are coupled through time, not only by the startup and shutdown costs, but also by minimum up and down time constraints.

The MIP problem formulation is an extension of the problem from Section 2.2. Additional sets of binary variables $\boldsymbol{u}, \mathbf{v}, \boldsymbol{w} \in \{0, 1\}$ are introduced, where $\boldsymbol{u}_{g,t}$ represents the on-line status of the generation unit g in time period t, while the binary startup and shutdown states are represented by $\mathbf{v}_{g,t}$ and $\boldsymbol{w}_{g,t}$ variables, respectively.

The constraints are either extended by the new binary variables, e.g., the injection limits (5d) are replaced by

$$\boldsymbol{u}_{g,t} \mathbf{p}_{\min}^{\mathrm{G}} \leq \mathbf{p}_{g,t}^{\mathrm{G}} \leq \boldsymbol{u}_{g,t} \mathbf{p}_{\max}^{\mathrm{G}}, \tag{7}$$

or the new constraints are added, such as the minimum up and down times of the dispatchable units

$$\sum_{n=t-\tau^{u}+1} \mathbf{v}_{g,n} \leq \boldsymbol{u}_{g,t}, \tag{8}$$

$$\sum_{n=t-\tau^{d}+1} \boldsymbol{w}_{g,n} \leq 1 - \boldsymbol{u}_{g,t}, \tag{9}$$

and a set of the constraints modeling startup and shutdown events

$$\boldsymbol{u}_{g,t} - \boldsymbol{u}_{g,t-1} = \mathbf{v}_{g,t} - \boldsymbol{w}_{g,t}. \tag{10}$$

The full UC model formulation is available in the MOST framework [30–32].

2.4. Swiss grid benchmark model

For the purpose of benchmarking the models introduced in the previous sections, two power models were set up. The LP ED problem was run using the proprietary model of the continental Europe, while the QP and MIP problems were applied to the model of the Swiss transmission grid introduced next. The topology of the grid, including the external nodes abroad, is illustrated in Fig. 2, consisting of 231 nodes and 439 transmission lines. The external nodes are used in order to model the imports and exports of the power. The load is evenly distributed across the nodes inside Switzerland up to a small random perturbation up to 100 MW with the net zero sum. The load data are 15 min samples collected by the national transmission system operator SwissGrid [34]. Fig. 1(a) illustrates the overall load during the first 15 days of January and June of 2020. The grid model also includes RES, namely, wind and solar energy, with the historical data shown in Figs. 1(b) and 1(c). The overall RES input is evenly distributed across the selected nodes in the network.

On top of the RES, the Swiss grid example also contains energy storage devices. For all LP models the length of the time period δt is set to 1 h, while in the QP and MIP simulations, the length of the time period δt is set to 15 min. The energy storage devices are located at the first N_{S} buses sorted according to the largest positive active load demand specified in the case file. The storage size $\boldsymbol{\epsilon}_{\mathrm{S}}^{\max}$ is chosen to contain up to 10 MWh. The initial state of charge is 70%, which represents $\boldsymbol{\epsilon}_0 = 0.7\boldsymbol{\epsilon}_{\mathrm{S}}^{\max}$. The storage device power ratings are limited to allow a complete discharging and charging within three hours and two hours, respectively. Therefore, $\mathbf{p}^{\mathrm{Sd},\max} = \frac{1}{3}\boldsymbol{\epsilon}_{\mathrm{S}}^{\max}$ and $\mathbf{p}^{\mathrm{Sc},\min} = -\frac{1}{2}\boldsymbol{\epsilon}_{\mathrm{S}}^{\max}$. All storage device discharging and charging efficiencies are chosen as $\eta_{\mathrm{d}} = 0.95$ and $\eta_{\mathrm{c}} = 0.93$.

3. Solution strategy

The resulting LP models are solved by both the DS and IP algorithms, while the QP and MIP models are solved by the IP algorithm. Historically, the DS was considered superior but this proposition has been challenged by advancements in IP methods, which outperform the DS, especially for large-scale problems. The computational complexity of the DS algorithm lies in the combinatorial nature of the search space defined by vertices of the feasible region, which grows very quickly for large LP problems. On the other hand, the computational bottleneck of the IP algorithm is the solution of a large sparse linear system in each IP iteration, which can be effectively mitigated by efficient direct sparse linear algebra routines [28,35]. When it comes to

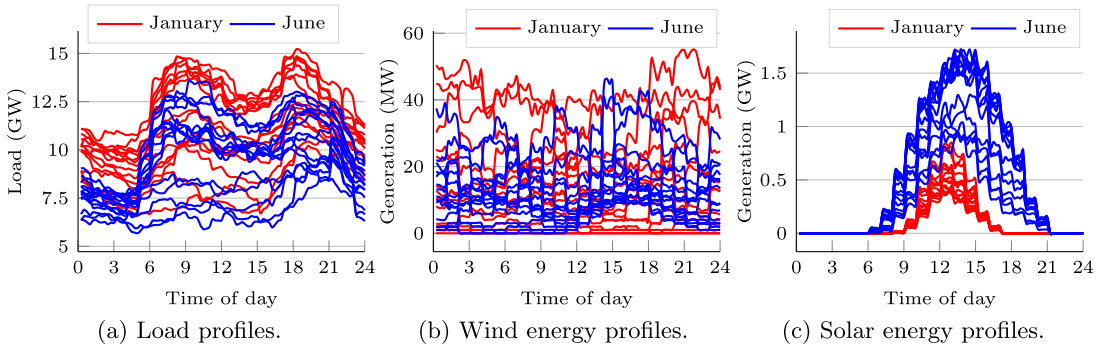


Fig. 1. Daily profiles of the Swiss grid benchmark example for selected months [33].

Fig. 2. Topology of the Swiss grid with the external nodes in the neighboring countries.

Table 1
Computational requirements of forecasting electricity prices on a typical day.

		Short-term forecasts	Long-term forecasts
(a)	Forecast horizon (days)	45	700
(b)	# of runs per day	15	1
(c)	# of weather scenarios	52	45
(d)	# of fuel & econ shocks	25	25
	# of optimizations per day[a]	68 850	66 500
	CPU time per optimization[b]	164	164
	CPU hours per day	3137	3029

[a] $a \times b \times (c + 2d)$.
[b] *Average time in seconds using the DS algorithm with the naive dispatch strategy.*

memory requirements, DS solves asymmetric linear systems of the size of the basis, which is much smaller compared to linear systems in the IP algorithms. Additionally, only a single column changes in every iteration, thus updating the factors is usually done rather than refactoring the whole matrix, which is recomputed only occasionally for numerical stability reasons. In the IP algorithm, the factorization is computed in every iteration and storing the factors requires a significant amount of memory [36]. The performance and memory requirements of both algorithms are compared in Section 5.

Additional challenges lie in processing the large number of scenarios associated with the highly dynamic nature of electricity markets. The LP models must be solved for a large number of equiprobable scenarios in order to get a reasonable estimation of the electricity prices distribution. This challenge is compounded by the fact that expectations of input variables are always changing. In the real-world trading environment, this means optimization of the entire problem set is done 15 times per day with the latest available input values. Table 1 shows the number of LP solves required per day in order to achieve acceptable prediction accuracy, which is 135 350 LPs at a cost of 6166 h of CPU time.

Given the large quantity of optimizations that must be completed on a continual basis, effective parallelization strategies are critical. Parallelization on the level of individual LPs is not considered, given its relatively small size. Instead, parallelization across the set of LPs offers far greater benefit. Thus, each LP solve is executed serially using a single CPU core. The main objectives of an effective computational strategy are optimizing the data pipeline such that the required data remain as close to the CPU registers as possible, and ensuring that when a core finishes processing its LP, there is another job assigned to it with minimal delay.

3.1. SLURM workload manager

SLURM Workload Manager is an open-source Linux utility that provides access to available computational resources for some duration of time required to perform computation in the context of heterogeneous multiuser, multinode clusters. SLURM is designed for scheduling massively parallel jobs, which usually take significant time to complete. If the program running time is small, on the order of less than one minute, and the number of scheduled tasks is very large, the SLURM scheduler will incur noticeable overhead. Such tasks should not be submitted as individual allocations, but rather packed into a single allocation containing multiple job steps. The disadvantage of this strategy is that it becomes difficult to keep all cores saturated with work when individual job steps finish. This difficulty stems from inadequate tools available in shell scripts and SLURM to detect job step completion, idle resources within an allocation, or specific dependencies between the individual job steps.

3.2. GREASY meta scheduler

GREASY is an open-source meta scheduler that works on top of SLURM to maximize resource utilization and minimize accounting overhead in embarrassingly parallel applications. GREASY is launched with a list of tasks to run, which are executed using the resources within a SLURM allocation. For each task, GREASY dispatches a job step if resources are available, otherwise it forms a queue and dispatches tasks as soon as resources become available. Additionally, users can control the compute resource utilization by adjusting the number of "workers" to adapt the scheduler to the character of various applications, whether memory or compute bound. A worker is an abstraction that GREASY uses to control the execution flow of job steps within the task list. Conceptually, a worker takes a task off the queue and runs it on the compute node such that each worker is always busy. The user may specify the number of workers in order to control how many processes will run concurrently on the allocated resources.

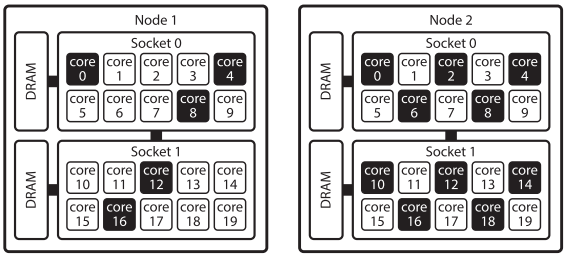


Fig. 3. GREASY CPU binding strategy for maximal scattering, Node 1 uses 5 workers while Node 2 uses 10.

As detailed in Section 5, undersubscribing nodes and executing on maximally scattered cores can substantially ease bottlenecks in memory bound applications.

The vanilla GREASY implementation dispatches job steps with minimal SLURM specifications, leveraging the process control of either SLURM or the underlying Unix kernel. The schedulers on the level of the operating system may shift processes to different cores during their lifetimes. This can improve throughput for CPU-bound applications, because while the process is waiting for input/output it gets evicted from the core so that another process can use the otherwise lost CPU cycles. Once the process is ready to continue, it is assigned to the next available core. For memory bound applications, however, this has dire performance consequences because data in cache memory is completely lost when a process is assigned to a different core. Scheduling processes such that they are executed on the same processor during their lifetime can dramatically improve performance by reducing the number of time-consuming cache misses.

We have extended GREASY such that it provides CPU affinity control [37], a feature that is not available in the original source code. An additional benefit of CPU affinity is that it also allows control over the load balance between the two CPU sockets. To achieve load balance, workers are spread out across maximally dispersed CPU cores based on CPU number as shown in Fig. 3.

4. Parallelism treatment

The stochastic scenarios representing different market developments are processed by running individual LP, QP, and MIP model instances (see Section 2) in parallel. Modern many-core CPUs allow multiple solves to be run simultaneously on a given compute node, however, this imposes greater congestion on the node's memory controller as the available bandwidth is shared across cores. Greater congestion in turn adversely impacts solution times.

Considering the increase in runtime of individual solves as the level of parallelism increases, it might be beneficial to reduce the amount of parallelism in favor of reducing the memory bottleneck. It is not obvious, however, how much the parallelism should be reduced in order to achieve the optimal hardware utilization. An empirical procedure proposed in this section may be used to determine the optimal level of parallelism. By following the procedure prior to the execution of subproblems on a given architecture, power grid practitioners can improve utilization of computational resources by avoiding the memory bottlenecks inherent in modern many-core CPUs. Consequently, significantly faster execution can thus be achieved for models comprising many independent subproblems.

In order to determine the optimal level of parallelism for a given model, it is important to find the average run time t^c_{avg} at each level of parallelism c. The model of interest is simply run multiple times, each time with a different number of instances running simultaneously, utilizing a different number of cores of the multicore CPU on a given compute node, as demonstrated in Fig. 4.

The performance of two different solution algorithms (DS and IP) on instances of the LP model are shown in Fig. 4(a). Given that the DS algorithm outperforms the IP algorithm across the entire range of solve concurrency, the choice between algorithms clearly favors the DS algorithm. This relationship between the number of concurrent solves and t^c_{avg} carries information not only about which algorithm will solve the problem faster, but also about the optimal level of parallelism with which to execute the given solution method. This difference can be mostly attributed to the difference in dynamic random access memory (DRAM) read volume. With $c = 128$ LPs running concurrently on the compute node, the IP executions will require a combined 7.2 TB of data from memory, while the DS executions will require only 1.7 TB. A similar measure is shown in Fig. 4(b) for the MIP and QP models.

The optimal level of parallelism, i.e. the number of concurrent model solves c, can be determined by finding the minimum expected processing time t_n of n model solves with respect to c. An approximation of this measure is given by

$$t_n \approx \frac{n}{c} t^c_{avg}. \tag{11}$$

The expected processing time t_n represents a lower bound for processing n model solves since it assumes no scheduling overhead or idle CPU time. However, since it depends on the experimental quantity t^c_{avg}, its numerical value bears some inherent error.

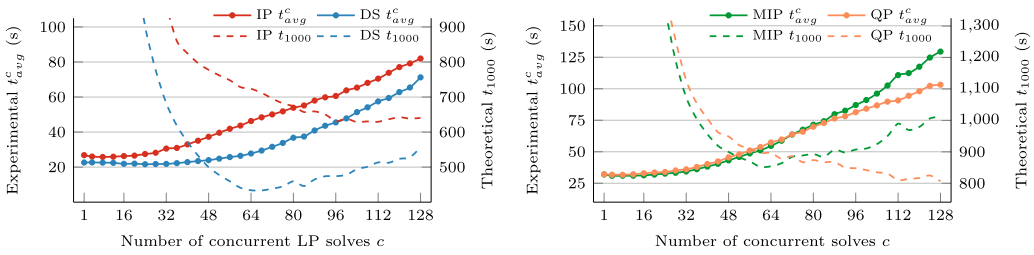
The theoretical processing times t_{1000} for a batch of $n = 1000$ jobs are shown in Fig. 4(a). The batch size n was set such that it reflects a realistic batch size encountered in practice. The point at which the minimum t_n is attained is relatively unaffected by changing batch size n except for very small batch sizes ($n < 320$) which do not arise in practice.

A substantial difference in the behavior of the algorithms can be observed with respect to parallelism, exhibiting two modes of model behavior. While the overall processing time t_{1000} using the IP algorithm reaches a minimum by exploiting the maximum level of parallelism, that is $c = 128$, with the DS algorithm the models are expected to be processed in the most efficient way by decreasing the level of parallelism to $c = 64$, utilizing only half of the available cores. A similar pattern is observed for the MIP and QP models in Fig. 4(b). While the QP model can efficiently utilize the maximum level of parallelism, the MIP model's performance improves by reducing the parallelism to $c = 64$.

Eq. (11) is an approximation of t_n, since it implies that partial batches, i.e., batches of solves that are less than the chosen level of parallelism, c, will be executed at a fraction of t^c_{avg} corresponding to the batch fraction. This would imply that a batch of 1 job, with $c = 128$, would be processed at $\frac{1}{128} * t^{128}_{avg}$ which is not the case. To eliminate this effect an alternative estimation of t_n is proposed:

$$t_n \approx \left\lceil \frac{n}{c} \right\rceil t^c_{avg}. \tag{12}$$

The ceiling function $\lceil \cdot \rceil$ implies that the compute node processes the jobs in batches of size c, and that the last batch must be processed at a cost of t^c_{avg}, irrespective of how many solves are remaining. This however represents a simplification on two accounts: (i) first it implicitly assumes that solve time is deterministic, so that each process in each batch of size c starts and ends at the same time. In reality, the solve time is a random variable, with a variance that increases as c increases. This suggests that as c increases, some processes will finish before others, and the cores will immediately start on a new job in the queue. (ii) Second, there is the observation that t^c_{avg} decreases



(a) LP model with IP and DS algorithms (b) MIP and QP models

Fig. 4. Average runtime t^c_{avg} as a function of c and expected processing time t_{1000} for a batch of $n = 1000$ problems.

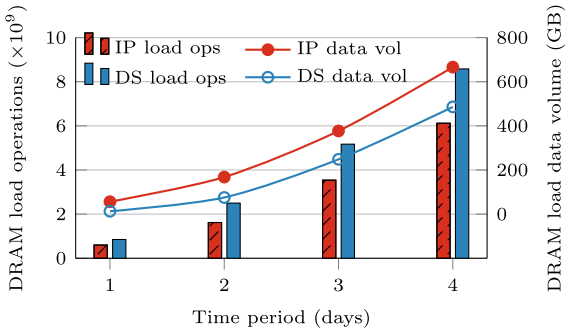


Fig. 5. IP and DS memory characteristics.

Table 2
Properties of the LP instances and data access characteristics of the solution algorithms.

Time period (days)	1	2	3	4
# of variables	71 120	113 792	156 464	199 136
# of constraints	104 480	167 376	229 721	292 116
Dual simplex algorithm				
Data load ops ($\times 10^9$)	38	118	246	422
L1 cache hit rate (%)	95.5	95.9	96.0	96.1
Cache miss rate (%)	2.3	2.1	2.1	2.0
Interior point algorithm				
Data load ops ($\times 10^9$)	32	87	190	346
L1 cache hit rate (%)	96.5	96.5	96.6	96.8
Cache miss rate (%)	1.9	1.9	1.9	1.8

as c decreases. Thus, when there are no remaining jobs in the queue, the effective c will rapidly decrease as it approaches 1, causing the remaining processes to speed up until all jobs finish. Given this, (11) gives a reasonable approximation for t_n from a theoretical perspective. The comparison of both predictions are evaluated and compared with actual measurements in the following section.

5. Numerical experiments

In this section, the computational nature of the power grid models is analyzed, focusing on the hardware–software interaction resulting from the application of optimization methods to solve the models. Multiple aspects of the solution process pipeline are analyzed, including (i) a choice of the solution algorithm based on its performance and impact on memory resources, and (ii) scheduling techniques responsible for allocation of computational resources to the individual model instances in massively parallel settings. The experiments were carried out on two Linux based compute clusters with different architectures: the 4-node "DXT Cluster", and the 41-node "ICS Cluster".[1]

In the numerical experiments, the performance of the solution methods is compared using the metric "solve time", which is defined as the wall time of the optimize function of the Mosek solver called for a single LP, QP, or MIP problem, which excludes the problem assembly and setup.

5.1. Performance analysis of the optimization algorithms

Performance and memory requirements of the DS and the IP algorithms are studied considering serial execution for LP models of increasing size, as shown in Table 2. First, the memory footprint of each algorithm is analyzed, since this imposes the main bottleneck in massively parallel simulations. Memory performance measurements are made using the LIKWID framework [38], accessing the performance counters on the Intel architecture.

From a memory perspective, Table 2 shows that the IP algorithm dispatches fewer data load requests (32 billion) compared to the DS algorithm (38 billion), and exhibits slightly better cache locality (1.9% miss rate) compared to DS (2.3%). As seen in Fig. 5, however, this economy of data load requests does not translate to economy of data transfers. Although fewer data load requests combined with a better overall cache hit rate results in a significant advantage in the number of DRAM load operations, these operations have far greater average data volume, resulting in a larger DRAM data transfer volume for the IP algorithm. This difference is most significant for the small problem size, where the IP algorithm reads 4.3× more data from DRAM (56 GB) than the DS algorithm (13 GB). These two examples serve as a baseline to show how the memory requirements can change significantly depending on the solution algorithm. Similarly, the memory footprint will be different for each model given differences in the problem structure.

To establish a baseline, solve times are measured with only a single model instance running in single-core mode on an otherwise idle compute node. Under these conditions, the solve time for a typical single-day LP is on the order of 20 s on either the DXT Cluster or the ICS Cluster. With these relatively small problem sizes and nonconcurrent execution, the speed difference between the algorithms is minimal. However, the IP algorithm exhibits superior scaling as the problem size grows in the multiday simulations, as seen in Fig. 6. The increase from 1 to 4 days results in a 7.8× increase in constraint matrix size. This causes an 8.7× increase in the solve time using the IP algorithm, compared to 16.5× for the DS algorithm. While the IP algorithm is clearly the better performer from a solve time perspective with large problem sizes, the advantage is less clear with the small problem size. With the introduction of parallelism in the following section, the superior memory performance of the DS algorithm will clearly shift the balance.

[1] The DXT Cluster uses nodes with two 64-core AMD EPYC 7702 1.5 GHz CPUs and 640 GB of memory, SLURM version 18.08.8, GREASY version 2.2.2, and MOSEK version 9.2.21. The ICS Cluster uses nodes with two 10-core Intel Xeon E5-2650 v3 CPUs and 64 GB of memory, SLURM version 20.02.4, GREASY version 2.2.2, and MOSEK version 9.2.29.

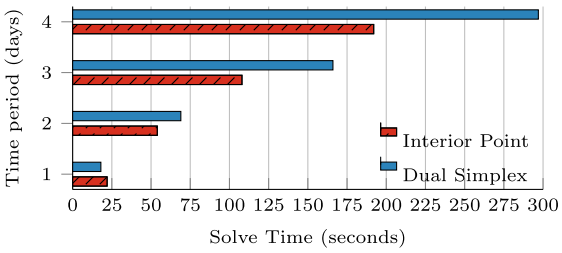


Fig. 6. IP and DS scaling for increasing problem size.

Table 3
Optimization problem sizes in terms of the number of variables and constraints. For MIP the continuous and integer variables are listed separately.

Grid model	Variables	Constraints
LP	71 120	104 480
MIP	57 429/3744	74 181
QP	180 117	213 717

5.2. Node-Level parallelism and memory bottleneck

The many-core CPUs of the DXT Cluster are exploited in order to run multiple model solves simultaneously on the given compute node. The optimization problems associated with the power grid models are summarized in Table 3. The memory bandwidth on the node becomes saturated and eventually congested as the degree of parallelism increases, since the available bandwidth is shared across the cores. This congestion adversely impacts the solution times, as shown in Fig. 7. All models and solution methods experience significant slowdown as the number of solves running concurrently increases (controlled by the number of GREASY workers). The mean slowdown when increasing from 32 to 128 concurrent solves ranges from 2.80× for the QP model, to 4.12× for the LP with the DS algorithm.

Another memory congestion indicator on the node is the runtime variance of individual processes. Fig. 7 shows how this variance increases substantially as the number of solves running concurrently increases. When increasing from 32 to 128 concurrent solves the standard deviation of runtime increases from 24.13× for the LP IP, to 46.31× for the MIP. This variance is caused by the memory controller scheduling the processes competing for the bandwidth as it becomes constrained. The substantial increase in variance indicates that memory bandwidth is relatively unconstrained for the lower level of concurrency, and severely constrained for the higher level of concurrency.

5.3. Analysis of parallelism modes

In order to better analyze different modes of the optimal parallelism determined by a procedure introduced in Section 4, derived metrics for the measured t^c_{avg} are provided in Fig. 8. In Fig. 8(a) the slowdown of t^c_{avg}, ψ is defined as

$$\psi(c) = \frac{t^c_{avg}}{t^1_{avg}}, \tag{13}$$

while in Fig. 8(b), the smoothed rate of change of t^c_{avg}, ϕ is defined as

$$\phi(c) = \frac{1}{6}\sum_{j\in K}\frac{t^j_{avg} - t^{j-4}_{avg}}{4}, \tag{14}$$

$$K = \{c - 4x \mid x \in \{0, \ldots, 5\}\}.$$

The processing time was observed to attain a minimum by either utilizing the full compute node with $c = 128$ or utilizing only half of the cores, $c = 64$, depending on the solution algorithm or the power grid model examined. The slowdown of t^c_{avg} with increasing level of parallelism relative to the single process execution t^1_{avg} is shown in Fig. 8(a). The two groups are visible by analyzing the rate of the slowdown, i.e., the slope of the slowdown curves, shown in Fig. 8(b). The LP with the DS algorithm and the MIP models both have generally increasing slopes throughout the entire range of c, while the slopes of the other two models stabilize around $c = 64$. This suggests that the slope is a more important factor than the total amount of slowdown since the LP with the IP algorithm and the QP both slow down a similar amount as the LP with the DS algorithm. The next subsection demonstrates how the predicted t_n translates to experimental results in massively parallel execution setup.

5.4. Massively parallel execution

Average runtime of c model solves running concurrently, t^c_{avg}, has been determined experimentally, as shown in Section 4. In the large-scale experiment considering $n = 1000$ model solves, both the LP model with DS and the MIP model are expected to achieve a minimum runtime t_{1000} at running 64 concurrent solves. On the other hand, the other two models are not expected to reach no decisive minimum before 128 concurrent solves, as shown in Fig. 4. Running the LP model with the DS algorithm with 64 concurrent processes results in an average solve time of 26 s and an average setup time of 5 s. Thus, expected processing time is

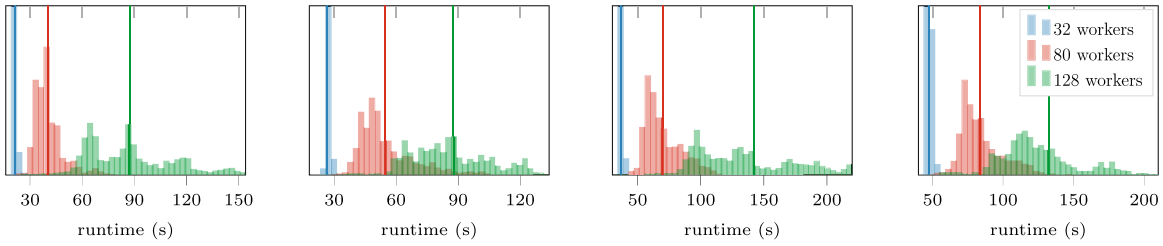
$$t^*_{1000} = \frac{1000}{64}(26 + 5)\text{ s} = 484\text{ s} \tag{15}$$

on a single compute node. To achieve such processing time, the assumption that all cores are fully utilized 100% of the time with no delays between individual LP solves running on a given core would have to be met.

Keeping all cores occupied is one responsibility of the workload manager such as SLURM. However, centralized schedulers such as SLURM are fundamentally designed for the traditional paradigm where there are a few large, long-running jobs, rather than ensembles of small, short-running tasks [39]. The strategy of submitting each individual model solve as a job batch to SLURM thus incurs extra overhead as the scheduler identifies and matches jobs to idle resources, accounts for user priority, prepares the environment, creates temporary directories, performs some sanity or health checks, etc. Considering the example with $n = 1000$ for the LP model with the DS algorithm, SLURM needs significantly more time to finish processing all LP solves, compared to the established expected processing time t^*_{1000}. The average processing time across 7 trials for SLURM is $t^{\text{slurm}}_{1000} = 787$ s seconds, as shown in Fig. 9a. These experimental results were obtained using a strategy that submits each LP solve to SLURM as a separate job batch, which maximizes the administrative overhead mentioned above. Multiple LP solves could be grouped into job batches, however, while such a strategy reduces administrative overhead, it becomes difficult to balance the batches to keep available CPUs fully saturated with jobs. Experimentally it was found that the administrative overhead of SLURM is much less than the idle CPU overhead that such job grouping strategies incur. To eliminate SLURM overhead while minimizing idle time across CPU cores, the meta scheduler GREASY is used.

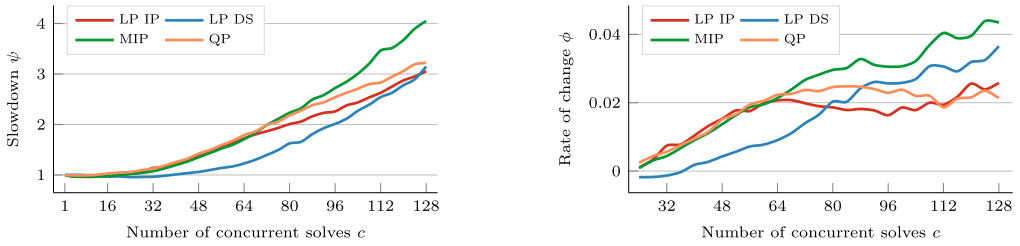
5.5. Massively parallel execution with meta scheduling

The primary goal of the meta scheduler is to eliminate the overhead incurred by SLURM. Considering a GREASY configuration using all 128 cores (128 workers), the execution of GREASY



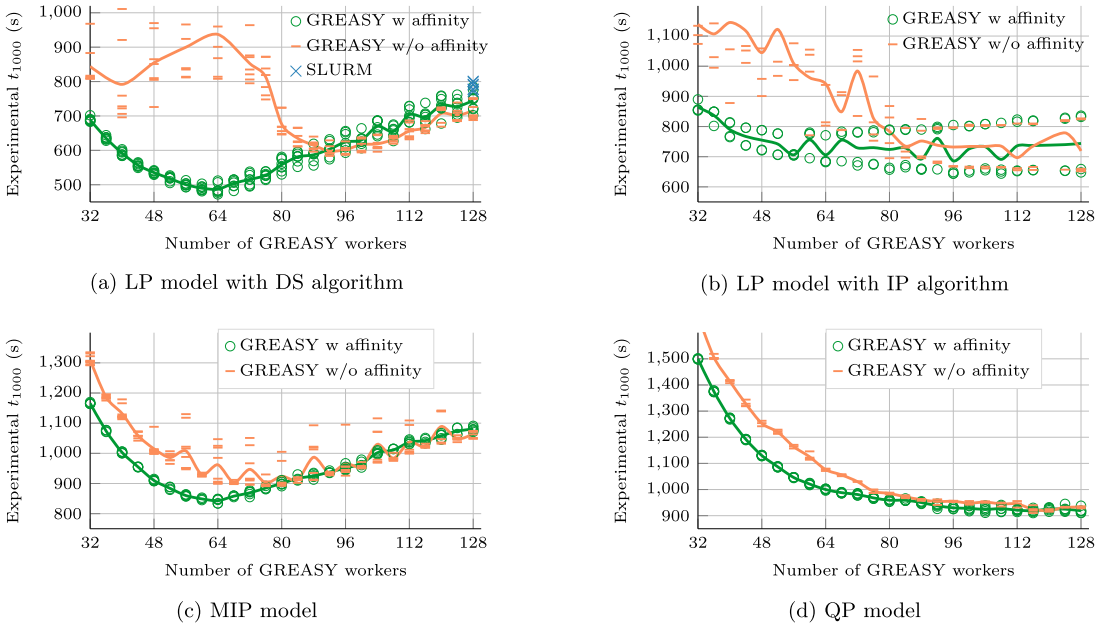
(a) LP model - DS alg. (b) LP model - IP alg. (c) MIP model (d) QP model

Fig. 7. Histograms of individual problem run times with CPU affinity ($n = 3000$), y-axis cropped at 1000. Vertical lines represent mean runtime.



(a) Slowdown of t^c_{avg} relative to single worker t^1 (b) Slowdown rate of change (average over 6 periods)

Fig. 8. Derived metrics for the average run time t^c_{avg} for all models.

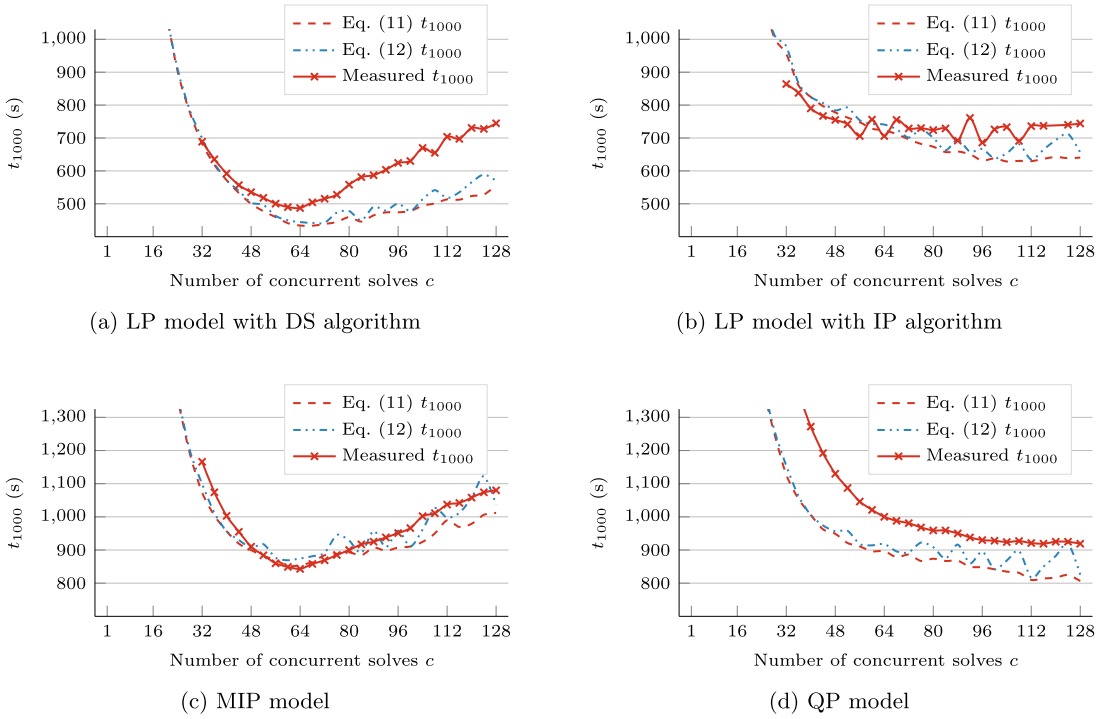


(a) LP model with DS algorithm (b) LP model with IP algorithm

(c) MIP model (d) QP model

Fig. 9. Performance of GREASY for various numbers of workers with and without CPU affinity control.

and SLURM are equivalent from a perspective of the number of solves that are running concurrently and the utilized resources. Fig. 9a illustrates that for the LP with the DS algorithm GREASY was able to achieve an average execution time $t^{\text{greasy}}_{1000} = 715$ s without CPU affinity control and $t^{\text{greasy}}_{1000} = 745$ s with CPU affinity control, which represents decrease of 9% and 5%, respectively, compared to t^{slurm}_{1000} on a single compute node of the DXT Cluster. The main benefit from using GREASY for data-intensive applications, however, arises from the ability to control the level of compute resources saturation. Since the memory bottleneck is exacerbated as the number of concurrent jobs increases, dispatching fewer concurrent solves to the compute node should benefit the overall processing time as established by (11). To control this parameter, GREASY uses the "worker" abstraction introduced in Section 3.2. Fig. 9 illustrates the effect of undersubscribing the DXT Cluster node consisting of 128 cores on a batch of 1000 model solves. These experimental measurements confirm the predictions of (11) that the optimal processing time $t^*_{1000} = 484$ s and thus optimal throughput should be achieved at 64 concurrent processes. The experimental measurement $t_{1000} = 487$ s differs from the predicted result by less than 1%, and represents a time



(a) LP model with DS algorithm

(b) LP model with IP algorithm

(c) MIP model

(d) QP model

Fig. 10. Predicted and experimentally measured processing time t_{1000} of 1000 LPs.

reduction of 38% compared to using the plain SLURM strategy, demonstrating how effective the reduction of parallelism is at reducing memory bottlenecks.

For models which are optimally executed with an undersubscribed node, controlling the CPU affinity is essential to achieving the optimal processing time, although the magnitude of the impact depends on the specific model. The comparison of experiments with and without such control is illustrated in Fig. 9. For a node employing at least 75% of the available cores, the difference is modest, but a gap of almost a factor of two occurs for the undersubscribed node of the LP DS, especially for the point at which the optimal throughput was established. This is the effect of the Unix kernel scheduler intervening with the affinity of the processes, effectively eliminating the benefit of data locality and thus losing the advantage brought by the cache hierarchy.

5.6. Verification of the optimal parallelism degree prediction

The correspondence between the experimental measurements of t_{1000} using GREASY, and the t_n predicted by (11) and (12) is discussed in this section. The two equations are compared against the experimental results obtained by running batches of 1000 jobs using GREASY with CPU affinity. The predicted values, introduced in Fig. 4, are qualitatively similar to the experimental values, as can be seen in Fig. 10. This demonstrates the fact that the predictions correspond well with experimental results in terms of determining the optimal level of parallelism that needs to be used in order to minimize the impact of memory bottlenecks. In this way, the processing throughput of the computational tasks is maximized.

The gap between the predicted and the measured t_{1000} may be attributed to further memory congestion introduced by the introduction of massive parallelism, as well as GREASY overhead. It can be also seen that the ceiling function in (12) introduces some nonlinearities to the predicted t_{1000} that do not appear consistently in the measured t_{1000}. However, the predictions do not change qualitatively using both formulations of the t_n prediction, thus the optimal level of parallelism can be found using either predictor.

6. Conclusions

For many time critical applications that consist of a large number of subproblems the usual approach to decrease computation time is to improve the solver. This, however, is often very expensive in terms of development time or license fees. Significant improvements can be achieved by optimizing the hardware utilization, either by selecting a method which reduces the bottlenecks or by adopting scheduling techniques better suited for the computational nature of the problem at hand. In the real-world trading environment, the careful management of the resources during the execution of embarrassingly parallel LP simulations improved the throughput of computations by 38%. This type of speedup is significant considering the computational demands of 135 350 optimizations per day, reducing daily computation time from 6166 CPU hours to about 2280 CPU hours.

This work proposed a simple procedure that can be used to determine the optimal level of node-level parallelism by power grid practitioners. The proposed procedure for parallel processing of a large number of simulations is based on a simple benchmark of empirical measurements. As such, it does not require any preliminary knowledge about the hardware architecture (e.g., the memory hierarchy properties) or characteristics of the solution algorithm (such as its memory access patterns). The limitation of this approach is such that the conclusions from one hardware architecture are not, in general, transferable to other architectures. The same applies to the particular problem at hand, where the conclusions are not transferable across the different problem types. Also the solution method needs to be considered, where various solution strategies might behave differently. Additionally, it is assumed that the problems included in the job pool are homogeneous from the computational perspective, i.e., having similar memory and time complexity, as well as the data access patterns. On top of this, the jobs are assumed to be independent, i.e., no interaction occurs between them.

Future work directions could focus on applying the proposed parallel computation schemes to additional problem instances

in the smart grid analysis, e.g., nonconvex nonlinear AC OPF. The methodology is production ready, and can be integrated in the tools such as GREASY, or other pilot job mechanisms as a preprocessing or analysis step suggesting to the user runtime parameters aiming to improve the hardware utilization. Additionally, before applying the procedure to the parallel decomposition schemes such as Benders, one should study how does the process synchronization (communication between the processes) impact the prediction.

CRediT authorship contribution statement

Juraj Kardoš: Conceptualization, Methodology, Software, Writing – original draft. **Timothy Holt:** Software, Investigation, Visualization, Writing – original draft. **Vincenzo Fazio:** Methodology, Software. **Luca Fabietti:** Methodology, Software. **Filippo Spazzini:** Conceptualization, Supervision. **Olaf Schenk:** Conceptualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] P. Fotis, S. Karkalakos, D. Asteriou, The relationship between energy demand and real GDP growth rate: The role of price asymmetries and spatial externalities within 34 countries across the globe, Energy Econ. 66 (2017) 69–84, http://dx.doi.org/10.1016/j.eneco.2017.05.027.

[2] C.G. Petra, A memory-distributed quasi-Newton solver for nonlinear programming problems with a small number of general constraints, J. Parallel Distrib. Comput. (2018) http://dx.doi.org/10.1016/j.jpdc.2018.10.009.

[3] C.G. Petra, O. Schenk, M. Anitescu, Real-time stochastic optimization of complex energy systems on high-performance computers, Comput. Sci. Eng. 16 (5) (2014) 32–42, http://dx.doi.org/10.1109/MCSE.2014.53.

[4] J. Kardoš, D. Kourounis, O. Schenk, Two-level parallel augmented schur complement interior-point algorithms for the solution of security constrained optimal power flow problems, IEEE Trans. Power Syst. 35 (2) (2020) 1340–1350, http://dx.doi.org/10.1109/TPWRS.2019.2942964.

[5] W. van Ackooij, I. Danti Lopez, A. Frangioni, F. Lacalandra, M. Tahanan, Large-scale unit commitment under uncertainty: an updated literature survey, Ann. Oper. Res. 271 (1) (2018) 11–85, http://dx.doi.org/10.1007/s10479-018-3003-z.

[6] M. Rahmani, S.H. Hosseinian, M. Abedi, Stochastic two-stage reliability-based security constrained unit commitment in smart grid environment, Sustain. Energy Grids Netw. 22 (2020) 100348, http://dx.doi.org/10.1016/j.segan.2020.100348.

[7] I. Gomes, R. Melicio, V. Mendes, Dust effect impact on PV in an aggregation with wind and thermal powers, Sustain. Energy Grids Netw. 22 (2020) 100359, http://dx.doi.org/10.1016/j.segan.2020.100359.

[8] F. García-Muñoz, F. Díaz-González, C. Corchero, A novel algorithm based on the combination of AC-OPF and GA for the optimal sizing and location of DERs into distribution networks, Sustain. Energy Grids Netw. 27 (2021) 100497, http://dx.doi.org/10.1016/j.segan.2021.100497.

[9] J.S. Guzmán-Feria, L.M. Castro, N. González-Cabrera, J. Tovar-Hernández, Security constrained OPF for AC/DC systems with power rescheduling by power plants and VSC stations, Sustain. Energy Grids Netw. 27 (2021) 100517, http://dx.doi.org/10.1016/j.segan.2021.100517.

[10] M.I. Alomoush, Microgrid dynamic combined power–heat economic-emission dispatch with deferrable loads and price-based energy storage elements and power exchange, Sustain. Energy Grids Netw. 26 (2021) 100479, http://dx.doi.org/10.1016/j.segan.2021.100479.

[11] G.A. Antonopoulos, S. Vitiello, G. Fulli, M. Masera, Nodal Pricing in the European Internal Electricity Market, Publications Office of the European Union, Luxembourg, 2020, http://dx.doi.org/10.2760/41018.

[12] M. Haberg, Fundamentals and recent developments in stochastic unit commitment, Int. J. Electr. Power Energy Syst. 109 (2019) 38–48, http://dx.doi.org/10.1016/j.ijepes.2019.01.037.

[13] M. Reolon Scuzziato, E. Cristian Finardi, A. Frangioni, Solving stochastic hydrothermal unit commitment with a new primal recovery technique based on Lagrangian solutions, Int. J. Electr. Power Energy Syst. 127 (2021) 106661, http://dx.doi.org/10.1016/j.ijepes.2020.106661.

[14] J. Kardoš, T. Holt, O. Schenk, V. Fazio, L. Fabietti, F. Spazzini, High-performance data analytics techniques for power markets simulation, in: 2021 International Conference on Smart Energy Systems and Technologies (SEST), 2021, pp. 1–6, http://dx.doi.org/10.1109/SEST50973.2021.9543110.

[15] J. Feinberg, H.P. Langtangen, Chaospy: An open source tool for designing methods of uncertainty quantification, J. Comput. Sci. 11 (2015) 46–57, http://dx.doi.org/10.1016/j.jocs.2015.08.008.

[16] A. Olivier, D.G. Giovanis, B. Aakash, M. Chauhan, L. Vandanapu, M.D. Shields, UQpy: A general purpose python package and development environment for uncertainty quantification, J. Comput. Sci. 47 (2020) 101204, http://dx.doi.org/10.1016/j.jocs.2020.101204.

[17] D. Suleimenova, H. Arabnejad, W.N. Edeling, et al., Tutorial applications for verification, validation and uncertainty quantification using VECMA toolkit, J. Comput. Sci. 53 (2021) 101402, http://dx.doi.org/10.1016/j.jocs.2021.101402.

[18] C. Hunziker, J. Lehmann, T. Keller, T. Heim, N. Schulz, Sustainability assessment of novel transformer technologies in distribution grid applications, Sustain. Energy Grids Netw. 21 (2020) 100314, http://dx.doi.org/10.1016/j.segan.2020.100314.

[19] I. Lampropoulos, T. Alskaif, J. Blom, W. van Sark, A framework for the provision of flexibility services at the transmission and distribution levels through aggregator companies, Sustain. Energy Grids Netw. 17 (2019) 100187, http://dx.doi.org/10.1016/j.segan.2018.100187.

[20] T. Kumamoto, H. Aki, M. Ishida, Provision of grid flexibility by distributed energy resources in residential dwellings using time-of-use pricing, Sustain. Energy Grids Netw. 23 (2020) 100385, http://dx.doi.org/10.1016/j.segan.2020.100385.

[21] A.B. Yoo, M.A. Jette, M. Grondona, SLURM: Simple linux utility for resource management, in: D. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 44–60.

[22] A. Jain, S.P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, K.A. Persson, FireWorks: a dynamic workflow system designed for high-throughput applications, Concurr. Comput.: Pract. Exper. 27 (17) (2015) 5037–5059, http://dx.doi.org/10.1002/cpe.3505, CPE-14-0307.R2.

[23] L.A. Wilson, J.M. Fonner, J. Allison, O. Esteban, H. Kenya, M. Lerner, Launcher: A simple tool for executing high throughput computing workloads, J. Open Source Softw. 2 (16) (2017) 289, http://dx.doi.org/10.21105/joss.00289.

[24] BSC Support Team, Greasy user guide, 2014, Version 2.14. URL: https://github.com/BSC-Support-Team/GREASY.

[25] N. Hanford, V. Ahuja, M. Farrens, D. Ghosal, M. Balman, E. Pouyoul, B. Tierney, Improving network performance on multicore systems: Impact of core affinities on high throughput flows, Future Gener. Comput. Syst. 56 (2016) 277–283, http://dx.doi.org/10.1016/j.future.2015.09.012.

[26] W. Xia, C.A. Shoemaker, Improving the speed of global parallel optimization on PDE models with processor affinity scheduling, Comput.-Aided Civ. Infrastruct. Eng. 37 (3) (2022) 279–299, http://dx.doi.org/10.1111/mice.12737, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/mice.12737.

[27] T. Kristiansen, The flow based market coupling arrangement in europe: Implications for traders, Energy Strategy Rev. 27 (2020) 100444, http://dx.doi.org/10.1016/j.esr.2019.100444.

[28] D. Kourounis, A. Fuchs, O. Schenk, Toward the next generation of multiperiod optimal power flow solvers, IEEE Trans. Power Syst. 33 (4) (2018) 4005–4014, http://dx.doi.org/10.1109/TPWRS.2017.2789187.

[29] T. Brijs, A. van Stiphout, S. Siddiqui, R. Belmans, Evaluating the role of electricity storage by considering short-term operation in long-term planning, Sustain. Energy Grids Netw. 10 (2017) 104–117, http://dx.doi.org/10.1016/j.segan.2017.04.002.

[30] C.E. Murillo-Sánchez, R.D. Zimmerman, C.L. Anderson, R.J. Thomas, Secure planning and operations of systems with stochastic sources, energy storage, and active demand, IEEE Trans. Smart Grid 4 (4) (2013) 2220–2229, http://dx.doi.org/10.1109/TSG.2013.2281001.

[31] R.D. Zimmerman, C.E. Murillo-Sanchez, R.J. Thomas, MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education, IEEE Trans. Power Syst. 26 (1) (2011) 12–19, http://dx.doi.org/10.1109/TPWRS.2010.2051168.

[32] R.D. Zimmerman, C.E. Murillo-Sánchez, MATPOWER optimal scheduling tool (MOST) user's manual, 2020, http://dx.doi.org/10.5281/zenodo.4073878.

[33] ENTSO-E, Statistics and data, 2021, (Accessed: 2021-12-01).
[34] Swissgrid, Production and consumption, 2021, URL https://www.swissgrid.ch/en/home/operation/grid-data/generation.html.
[35] M. Bollhöfer, O. Schenk, R. Janalik, S. Hamm, K. Gullapalli, in: A. Grama, A.H. Sameh (Eds.), Parallel Algorithms in Computational Science and Engineering, Springer International Publishing, 2020, pp. 3–33, http://dx.doi.org/10.1007/978-3-030-43736-7_1.
[36] J. Kardoš, D. Kourounis, O. Schenk, in: A. Grama, A.H. Sameh (Eds.), Parallel Algorithms in Computational Science and Engineering, Springer International Publishing, 2020, pp. 63–93, http://dx.doi.org/10.1007/978-3-030-43736-7_3.
[37] X. Abellan, J. Naranjo, C. Simarro, J. Rodriguez, P. Rodenas, T. Holt, GREASY: CPU affinity extension, 2021, http://dx.doi.org/10.5281/zenodo.4668583.
[38] J. Treibig, G. Hager, G. Wellein, LIKWID: A lightweight performance-oriented tool suite for X86 multicore environments, in: Proceedings of the 39th International Conference on Parallel Processing Workshops, IEEE Computer Society, 2010, pp. 207–216, http://dx.doi.org/10.1109/ICPPW.2010.38.
[39] D.H. Ahn, N. Bass, et al., Flux: Overcoming scheduling challenges for exascale workflows, Future Gener. Comput. Syst. 110 (2020) 202–213, http://dx.doi.org/10.1016/j.future.2020.04.006.