Hardware–software codesign for peer-to-peer energy market resolution

Beatrice Thomas, Roman Le Goff Latimier, Hamid Ben Ahmed, Gurvan Jodin, Abdelhafid El Ouardi, Samir Bouaziz

▶ To cite this version:

Beatrice Thomas, Roman Le Goff Latimier, Hamid Ben Ahmed, Gurvan Jodin, Abdelhafid El Ouardi, et al.. Hardware–software codesign for peer-to-peer energy market resolution. Sustainable Energy, Grids and Networks, 2023, 35, pp.101122. 10.1016/j.segan.2023.101122 . hal-04202817

HAL Id: hal-04202817
https://hal.science/hal-04202817v1

Submitted on 4 Oct 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Hardware-Software Codesign for Peer-to-Peer Energy Market Resolution

Beatrice THOMAS[a,b,*], Roman LE GOFF LATIMIER[b], Hamid BEN AHMED[b], Gurvan JODIN[b], Abdelhafid EL OUARDI[a], Samir BOUAZIZ[a]

[a] *Universite Paris-Saclay, ENS Paris-Saclay, CNRS, SATIE, Gif-sur-Yvette, 91190, France*
[b] *SATIE, ENS Rennes, CNRS, Bruz, France, Bruz, 35170, France*

Abstract

The growth of distributed energy resources raises the challenge of scaling up network management algorithms. This difficulty may be overcome in operating conditions with the help of a rich literature that frequently calls upon the distribution of computations. However, this issue persists during preliminary simulations validating the performances, the operation's safety, and the infrastructure's sizing. A hardware-software co-design approach is conducted here for a Peer-to-Peer market to address this scaling issue while computing simulations on a single machine. The mapping between several algorithms and different models of partitioning on Central and Graphic Processing Units (CPU-GPU) has been conducted. The complexity and performance of the Operator Splitting Quadratic Program (OSQP) or Alternating Direction Method of Multipliers (ADMM) for a centralized or decentralized resolution according to the hardware have been studied and analyzed in different test cases. The dominance of the pair ADMM and GPU has been demonstrated by having a speed-up of more than 98% compared to the other methods when the market has more than 500 agents.

Keywords: Electricity market, Peer to Peer (P2P), CPU-GPU Partitioning, Parallel computing, Performance evaluation

Nomenclature

$\#(.)$ Cardinality of a set

$\Lambda = (\lambda_{nm})$ Matrix of trade prices λ_{nm} of agent n with agent m

$g_n(P_n) = a_n \cdot P_n^2 + b_n \cdot P_n$ Cost function of agent n

$N = |\Omega|$ Total number of agents

Ω Agent's Community

Ω_c Set of consumers (subset of Ω)

Ω_g Set of generators (subset of Ω)

ω_n Set of partners of agent n

Ω_p Set of prosumers (subset of Ω)

$\underline{p_n}/\overline{p_n}$ Lower/upper power boundary of agent n

lb_n/ub_n Lower/upper trade boundary of agent n

$M = \sum_{n\in\Omega} M_n$ Entire number of peers (possible trades count)

$M_n = \#(\omega_n)$ Peers count of the agent n

$P = (p_n)$ Vector of the total amount of power p_n traded by agent n

$T = (t_{nm})$ Matrix of trade powers t_{nm} of agent n with agent m

1. Introduction

To achieve carbon neutrality by 2050 for electricity generation, the power grid must be prepared for the massive integration of Distributed Energy Resources (DER) [1]. These include renewable energies, mainly distributed producers, who will significantly impact the grid [2]. Furthermore, many so-called active consumers (smart home [3], electric vehicle, battery, load management) will be more and more present on the grid. The coordination of all these new agents calls for a modification of

*Corresponding author : beatrice.thomas@ens-rennes.fr, École normale supérieure de Rennes Campus de Ker Lann 11, avenue Robert Schuman 35170 BRUZ - France

the mechanisms of power systems management. Indeed, solving a management problem with such a large number of actors gets more and more difficult and certainly impossible in a centralized approach due to its algorithmic complexity [4].

Up to now, electricity markets have proven their operational aptitude to handle various agents aiming at maximizing their social welfare while balancing production and consumption offers. This balance is crucial to prevent any collapse of the entire grid. Therefore, several markets operate in succession, prior but closer and closer to the delivery time, to adjust the balance as much as possible as the forecasting uncertainties are reduced. Nowadays, a centralized market agent operates marketplaces, gathering offers and performing market clearing. But with the increase of the agent number, this central agent may face both a communication bottleneck and a computational challenge [5].

To address this limitation, decentralized approaches have resulted in an extensive literature. The large-scale management problem is then divided into a multitude of local problems that are coordinated with each other [5]. While reducing the computational load of the agents, it relies on exchanging messages between several machines through a communication network [6]. Among other architectures of the distributed resolution, a Peer to Peer (P2P) market is fully decentralized since there is no longer a central agent but only communications between peers. It allows the introduction of specific features, such as heterogeneous preferences. Besides being a market structure that can be used, the P2P market can be considered as a formal generalization of any other network: centralized, hierarchical [7] ...

Decentralized mechanisms enable resolution algorithms where each agent performs a part of the computation, making real-time resolution possible with a manageable computational cost for each agent. However, before being implemented in the actual grid, some challenges must be addressed by the P2P market[8]. Thus, preliminary simulations - both from systems operators and researchers- are necessary for numerous purposes: designing the rules of this new market, assessing its performances, guaranteeing its safety, and anticipating new infrastructures. These simulations must be done on realistic study cases, *i.e.*, large-dimension cases, to observe the algorithm's scale effects [9], and over a long period to keep temporal coherency while verifying the behavior during extreme events [10]. As these simulations are performed upstream of the actual deployment, they cannot yet involve the distributed computing capacities of the agents that will be involved in the future. These simulations being made centralized on a unique hardware target, prohibitive processing time resurfaces with the increase in agents. Studies on real-size systems then become unfeasible. This observation is not limited to market problems but is also valid for network simulations as power flows (PF) [11]-[14], or for combined resolution as optimal power flows (OPF) [15]-[16].

Tab. 1 presents an illustrative state of the art of computation time on the different electric grid problems depending on the number of agents and the target hardware architecture. The purpose here is not to be exhaustive or to compare times between studies, as they are different problems that cannot be compared. But to illustrate the above observation based on several independent studies. Readers must be aware that 9s may appear small, but this is the time for one step and 30-agents case. The time to process a 3000-agents test case will be at least 10000 times higher (as the complexity is at least quadratic). One day to compute one step is too slow to enable the simulation over a large period with, for example, one hour step. In addition, computation time and hardware used for simulation are not systematically indicated in the publications. When it is not specified otherwise, the time is for the simulation of the complete resolution. It is worth noticing that [17] is a market resolution that also considers the grid status. Readers' attention is also drawn to the comparison of several algorithms and implementations done in [15]. It results in a variety of computation times for similar problems. It clearly highlights the importance of carefully selecting the method and the hardware target to reduce the simulation times.

In all cases, considerable computation times can be observed when the problem size increases. This empirical increase is sharper than a linear evolution according to the number of

Table 1: Comparison of different electrical problems and their computation times on CPU

Problem	hardware target	size (N)	time
OPF decentralized [15]	CPU	48	$> 0,3s$ $< 483s$
PF decentralized [11] & communication delay	CPU	34	9.4s
decentralized market [18]	CPU	31	9.5s
decentralized market [17]	CPU I5-7200U	31	59s
decentralized market [19]	CPU I7-6500U	12	0.1s
centralized MASCEM[20]	CPU	12 1446	4.4s 65s
centralized PF [13]	CPU Xeon E5-2620	14 9241	0,115s 197s
centralized OPF [16]	CPU E5-2650	30 300	212.9s 37 000s

agents. Moreover, when possible, the parallelization of tasks across the cores of a CPU allows, at most, a reduction by a factor equal to the number of cores, although this limit generally cannot be reached. Therefore, the design of distributed management mechanisms is confronted with a computational complexity bottleneck that makes any simulation of a large-scale real system unfeasible.

Beyond the use of parallelization, this lock calls for a Hardware-Software Codesign approach in order to be able to carry out the large-scale simulations required for the deployment of new power system management regulation. This approach consists in finding the best pair algorithm architecture for one application, the P2P market in the present study. Due to the decentralized approach of this application, the massively parallel architecture, such as General-Purpose Graphics Processing Units (GP-GPU), will preferably be explored as it gives access to considerable computing power as long as the tasks to be performed match the internal architecture of the component. Indeed, this type of computing architecture has already proven its performance in various domains: radars [21], image processing [22], and cartography [23]. Even within the power system field, GP-GPU has also been used to reduce simulation time on Power Flow problems [12]-[14] and Optimum Power Flow [16]. As an example, two methods and their performance on CPU-GPU architectures are resumed in Tab. 2.

Table 2: Comparison of the different electrical problems and their computation times on CPU-GPU

Problem	hardware target	size (N)	time
centralized PF [13]	CPU-GPU Xeon E5-2620 NVIDIA Tesla K20c	14 9241	1,528s 86,46s
centralized OPF [16]	CPU-GPU Xeon E5-2620 NVIDIA Tesla K20c	30 300	13.1s 2 300s

By following Hardware Software Codesign approach, the present work aims to allow the P2P market resolution in the case of a massive number of agents. This will make it possible to simulate large systems over an extended period. The main contributions of this work are the following:

- comparison of the computational and memory complexity of several algorithms of a P2P market resolution;
- different implementations with a CPU-GPU partitioning, for each algorithm ;
- an optimized Alternating Direction Method of Multipliers (ADMM) formulation targeted for massively parallel computing;
- performance evaluation on a European data set and on random cases.

The remainder of this paper is structured as follows: first, part 2 presents the market problem and analyzes two algorithms for the minimization resolution. Then part 3 will focus on describing all the optimizations made to achieve a CPU-GPU partitioning of the algorithms. Finally, section 4 will present the obtained results following hardware-software co-design approach.

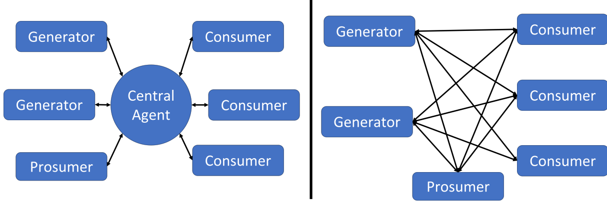

Figure 1: Network for a centralized (left) or a decentralized (right) market [24]

2. Peer-to-peer market and resolution approaches

2.1. Centralized resolution: overview and definitions

To integrate a large number of participants in the operation of power systems, peer-to-peer markets are a frequently investigated paradigm. As illustrated Fig. 1, a central agent is no longer required in the market. This decentralized operation allows scaling the number of agents. Rather than a power to exchange with the network, each peer negotiates an exchange with another peer. This allows the introduction of product differentiation: a peer can have heterogeneous preferences between its different partners.

Besides its direct application as such, this form of market is a generalization of different communication structures: for instance, a centralized market can be considered as a peer-to-peer market where one peer is connected to all the others. This justifies its choice for this study. The considered market (1) aims at minimizing the cost functions of all agents: generators $n \in \Omega_g$, consumers $n \in \Omega_c$ and prosumers $n \in \Omega_p$, and their product differentiation γ_{nm}.

$$\min_{T,P} \sum_{n\in\Omega} \left(g_n(p_n) + \sum_{m} \gamma_{nm} t_{nm} \right) \tag{1a}$$

$$\text{subject to} \quad t_{nm} = -t_{mn} \qquad (\lambda_{nm}) \qquad n,m \in \Omega \tag{1b}$$

$$p_n = \sum_{m\in\omega_n} t_{nm} \qquad (\mu_n) \qquad n \in \Omega \tag{1c}$$

$$\underline{p_n} \leq p_n \leq \overline{p_n} \qquad n \in \Omega \tag{1d}$$

$$t_{nm} \leq 0 \qquad n \in \Omega_c \tag{1e}$$

$$t_{nm} \geq 0 \qquad n \in \Omega_g \tag{1f}$$

$$\underline{p_n} \leq t_{nm} \leq \overline{p_n} \qquad n \in \Omega_p \tag{1g}$$

The production and flexibility costs are assumed to be expressed as convex functions, bounded by the limits (1d). The power exchanged with the network by an agent is equal to the sum of all its trades (1c) with its partners ω_n, a constraint associated with the dual variable μ_n. In order to maintain the balance between production and consumption, each trade is submitted to a reciprocity constraint (1b) associated with λ_{nm}.

In an actual application, each peer n is connected to a set of partners ω_n; hence the number of trades is proportional to N, the number of agents. However, the extreme case of a completely connected market where $\#(\omega_n) = N - 1$ would generate $(N-1)^2$ trades. This would lead to a quadratic program involving $N_v = O(N^2)$ variables and subject to $N_c = O(N^2)$ linear constraints. To solve this problem, a resolution algorithm will be characterized by its algorithmic complexity (number of operations) and its spatial complexity (used memory space). Those parameters will determine how algorithms scale with the number of agents.

In order to assess the resolution complexity of this problem, an optimization algorithm must be selected. OSQP[1] (Operator Splitting Quadratic Program) has been chosen here since it is considered a state-of-the-art reference algorithm for quadratic problems [25]. Appendix 6 provides its computational complexity analysis. In this fully connected market, the computational complexity is in $O(N^{2*\alpha})$ with $\alpha > 2$ as it can be seen in Fig. 2. Only one problem must be stored, so the spacial complexity is about $O(N^4)$ and can be reduced to $O(N^3)$ thanks to sparsity.

A 250 s delay to solve a 200-agents market makes it clear that the resolution of a P2P market cannot scale in a centralized manner. Solving the P2P market thus raises a computational complexity barrier in its centralized form. Consequently, the simulation of large-scale markets is compromised at this stage.

[1] https://github.com/oxfordcontrol/osqp

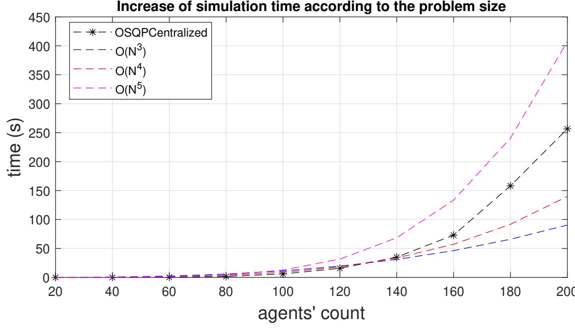

Figure 2: Measured computational time (black) and trend curves according to the problem's size for an OSQP centralized resolution (C++ on CPU, AMD RYZEN 5 5600H)

2.2. Decentralized peer-to-peer market algorithm

Considering the computational complexity of the centralized resolution of the P2P market, the following section focuses on the description of decentralized algorithms, along with the estimation of their algorithmic complexity. Using the Alternating Direction Method of Multipliers (ADMM), [26], (1a) and (1b) can be re-written for each agent n, and with several steps k with ρ the penalty factor (see Appendix 6 for more details):

$$
\begin{aligned}
T_n^{k+1} = \underset{T_n}{\operatorname{argmin}} \quad & g_n(p_n) + \sum_{m\in\omega_n} \gamma_{nm} t_{nm} \\
& + \sum_{m\in\omega_n} \Big(\frac{\rho}{2}(t_{nm} - \frac{t_{nm}^k - t_{mn}^k}{2} + \frac{\lambda_{nm}^k}{\rho})^2\Big) \\
\text{s.t.} \quad & t_{nm} \in C
\end{aligned} \tag{2}
$$

T_n being the vector containing the agent (n)'s trade offers. In this expression, C is the convex set gathering (1c)-(1g) constraints. In each step, the Lagrangian multipliers are updated to ensure the anti-symmetry of the exchanges.

$$
\lambda_{nm}^{k+1} = \lambda_{nm}^k + \frac{\rho}{2}(t_{nm}^{k+1} + t_{mn}^{k+1}) \tag{3}
$$

Primal (symmetries) and dual residuals (changes between each step) are evaluated as follows:

$$
\begin{aligned}
r_n^{k+1} &= \sum_{m\in\omega_n} (t_{nm}^{k+1} + t_{mn}^{k+1})^2 \\
s_n^{k+1} &= \sum_{m\in\omega_n} (t_{nm}^{k+1} - t_{nm}^{k})^2
\end{aligned} \tag{4}
$$

This algorithm is highly parallel as each agent can perform his own minimization independently. Results still need to be broadcast to evaluate the residuals and reach the consensus.

```
   input  : Study case, ρ, ϵ and kmax
   output: T, Λ, Residuals
1  while err > ϵ and k < kmax do
2      for n=1,...N do                          // agents
3          T_n^{k+1} ← (2);
4      end
5      for n=1,...N do                          // agents
6          for j=1,...m do                      // peers
7              r_n^{k+1}, s_n^{k+1} ← (4) ;
8              Λ_nm ← (3) ;
9          end
10     end
11     Res_R ← sum(r_n^{k+1}) ;
12     Res_S ← sum(s_n^{k+1}) ;
13     err ← max(Res_R, Res_S) ;
14 end
```

Algorithm 1: Global algorithm

2.3. Global resolution complexity

To solve the global problem described by Alg. 1, the local problem (2) must be solved N times (one per agent) per iteration. The other operations (residuals calculation, Lagrangian multiplier update...) are the same for each agent and have a complexity of $O(N^2)$. With C_l^n being the complexity of the local problem (2) for agent n, the total algorithm's complexity will be:

$$
C_g = O\Big(N^2 + \sum_{n<N} C_l^n\Big) \tag{5}
$$

Similarly, to store the trade matrix T or the dual variable Λ, the memory used is at least $O(N^2)$. This is smaller than the centralized complexity of $O(N^4)$. If the agents are clustered by k, the complexity computation became :

$$
C_g(k) = O\Big(N^2 + \sum_{i<N/k} C_l^i(k)\Big) \tag{6}
$$

This algorithm has anticipated ending conditions that cannot be known beforehand. Hence the scaling of the average number of iterations cannot be calculated a priori. Therefore, only the worst-case scenario complexity will be checked: the anticipated ending conditions are never used. The following paragraphs will focus on evaluating the complexity of the local problem

resolution C_l^n according to two different and representative approaches. To do so, it is necessary to provide an expression for the cost functions. They will be here supposed to be quadratic [27] as follows:

$$g_n(p_n) = 0,5 \cdot a_n \cdot p_n^2 + b_n \cdot p_n \tag{7}$$

The flexibility of a consumer will then be indicated by a minimum cost for the expected power and a stiffness a_n. The local problem becomes a linear constrained quadratic program. Thus, the first option will be a direct OSQP resolution [25]. To harness the local problem parallelization, the second option will again be a decomposition by ADMM.

Local minimization by OSQP. In a fully connected market, N_c and N_v are $O(N)$ for the local minimisation. Thus, $C_l^n = O(N^\alpha)$ and the global computational complexity will be $C_g = O(N^{\alpha+1})$. To use the OSQP algorithm, at least N matrices with a $O(N^2)$ size must be stored (one matrix A for each agent). Therefore, the spatial complexity is about $O(N^3)$. By considering p_n as a variable for each agent in the local minimization and the constraint (1c) in A, the sparsity of all matrix reduces the spacial complexity at $O(N^2)$. By clustering the agents, the system to solve is about $(k \cdot N)$ sized; thus, the complexity will be about $C_l^i(k) = O(k^\alpha N^\alpha)$. The global computational complexity will then be $C_g(k) = O(k^{\alpha-1} N^{\alpha+1})$. So clustering the agents is not attractive as it significantly worsens the complexity. With OSQP solving local problems and in the worst-case scenario (when all agents are solved together, $k = N$), the complexity of the global algorithm is $C_g(N) = O(N^{2\alpha})$ as in the centralized computation.

Local minimization by sharing ADMM. The second presented approach focuses on the parallelism exploitation in the local minimization (2) by formulating it as a sharing problem (see Appendix 6 for more details). According to [26], let k be the global step, j the local step, t_i the ith t vector's component corresponding to the ith trade of the considered agent, $\bar{t}$ the mean value of the trades of agent n, and finally $\tilde{p} \approx \frac{p_n}{M_n}$. The solution (where $\tilde{p} = \bar{t}$) is reached by iterating the following steps for each agent n:

$$t_i^{j+1} = \underset{lb_n \le t_i \le ub_n}{\text{argmin}} \left(f_i(t_i) + \frac{\rho_l}{2} \left\| t_i - t_i^j + \bar{t}^j - \tilde{p}^j + \mu^j \right\|_2^2 \right) \tag{8a}$$

$$\tilde{p}^{j+1} = \underset{\underline{p_n} \le M_n \tilde{p} \le \overline{p_n}}{\text{argmin}} \left(g(M_n \tilde{p}) + \frac{M_n \rho_l}{2} \left\| \tilde{p} - \mu^j - \bar{t}^{j+1} \right\|_2^2 \right) \tag{8b}$$

$$\mu^{j+1} = \mu^j + \bar{t}^{j+1} - \tilde{p}^{j+1} \tag{8c}$$

with lb_n and ub_n the boundaries on the trade (1e)-(1g) and:

$$f_i(t_i) = \frac{\rho}{2}\left(t_i - \frac{t_{ni}^k - t_{in}^k}{2} + \frac{\lambda_{ni}^k}{\rho}\right)^2 + \gamma_{ni} t_i \tag{9}$$

All the functions that must be minimized are scalar and quadratic, so their minima can be expressed analytically. These functions can be noted as follows:

$$\sum_l a_l \cdot (y - b_l)^2 + \sum_l c_l \cdot y \tag{10}$$

a and c coefficients are constants. Only the b coefficients will change according to the global iteration k or the local iteration j. Trades update (8a) coefficients are noted b_{t1} and b_{t2}, power update (8b) coefficient is noted b_{p1}:

$$b_{t1} = 0.5(t_{ni}^k - t_{in}^k) - \frac{\lambda_{ni}^k}{\rho} \tag{11a}$$

$$b_{t2} = t_i^j - \bar{t}^j + \tilde{p}^j - \mu^j \tag{11b}$$

$$b_{p1} = \mu^j + \bar{t}^{j+1} \tag{11c}$$

To check the algorithm convergence and have an early termination condition, residuals can be computed:

$$\begin{aligned} r^{j+1} &= \left\| \bar{t}^{j+1} - \tilde{p}^{j+1} \right\| \\ s^{j+1} &= \left\| t_i^{j+1} - t_i^j \right\| \end{aligned} \tag{12}$$

The local algorithm for each agent is resumed in Alg. 2.

In this local algorithm, finding the minimum needs a constant number of operations at each iteration for each peer. Thus with a fully connected market, the local complexity is $C_l^n = O(\#(\omega_n)) = O(N)$. In this worst case the global complexity then becomes $C_g = \sum_{n<N} \#(\omega_n) = M = O(N^2)$. Furthermore, several $N \cdot N$ matrices whose number does not depend on

```
input  : j_max, ub_n, lb_n, peers
output: t_min
1  while err > ϵ and j < j_max do
2  |  for i=1,...M do                         // all peers
3  |  |  t_i^{j+1} ← (8a);
4  |  |  t_i^{j+1} ← max(min(t_i^{j+1}, ub_n), lb_n) ;
5  |  end
6  |  t̄^{j+1} ← mean(t^{j+1}) ;
7  |  p̃^{j+1} ← (8b) ;
8  |  p̃^{j+1} ← max(min(p̃^{j+1}, p̄_n/M_n), p_n/M_n) ;
9  |  μ^{j+1} ← (8c) ;
10 |  (s^{j+1}, r^{j+1}) ← (12) ;
11 |  err ← max(s^{j+1}, r^{j+1}) ;
12 end
13 t_min ← t^j ;
```
Algorithm 2: ADMM local algorithm

the agents' count are stored. Thus, the spatial complexity is at most $O(N^2)$.

When the agents are clustered, the local algorithm's complexity is $C_l^i(k) = O(\sum_k M_k \simeq k \cdot N)$ for each cluster. There are N/k clusters, so the total complexity remains $O(N^2)$. Thus, agents can be clustered and vector concatenated without worsening the computational complexity. Furthermore, the memory and computational complexity are $O(N^2)$, which is the best possible for this total algorithm.

The complexity analysis presented here is built on the radical assumption that each peer is connected to all the others; hence the number of trades is in N^2. This will hardly ever be the case during a real deployment where the maximum number of peers will likely be fixed to a constant value. The announced complexity values would then all be reduced by one exponent. Nevertheless, a fully connected market ensures that the decentralized algorithm will converge to the same value as the centralized one and, thus, is initially required for the simulation. The problem being strictly convex, with the penalty factors tuned, the algorithm is guaranteed to converge to the optimum value.

The previous sections have highlighted that the resolution of a P2P market presents a complexity that strongly scales with the number of agents. For the ADMM the computational complexity is the same as the memory complexity ($O(N^2)$), whereas the computation is dominant for OSQP ($O(N^{\alpha+1})$ against $O(N^3)$). Not only this means that the appropriate hardware must be selected according to the operational intensity: the number of operations divided by the memory used. But also, particular attention must be paid to the implemented algorithm to make the most of the hardware architecture.

3. Hardware-Software Codesign for P2P Energy Market Resolution

The hardware-software codesign approach consists of carrying out an algorithm-architecture mapping in order to identify an architecture model that meets the requirements of computational complexity and processing times. To do so, a CPU-GPU partitioning and an algorithm's optimization for all algorithms are presented in this section. This consists in allocating specific tasks between the CPU and GPU to optimize the overall performance. The original algorithm should be modified to make the most of this approach to ensure that computations are suited for the GPU regarding parallelization and memory access.

All operations must not be deported onto the GPU to optimize the processing time. For instance, in [10]-[11], only specific operations have been deported (such as the Jacobian matrix calculation) onto the GPU to reduce the processing time. Indeed, the CPU is flexible and versatile, with small latency compared to GPU. The GPU can significantly reduce computation time if the data processing architecture is designed with a highly parallel model and efficient memory management. This high parallelization is only possible by considering the GPU-specific hardware features. For instance, all threads within a warp (smallest schedulable unit on a GPU) must compute the same work; if not, their execution will be serialized [28]. To do so, it is important to do branchless programming on the threads. Suppose their code asks them to compute the same instruction. In that case -the GPU is a Single Instruction Multiple Data hardware (SIMD)- all threads within a warp will materially proceed to the same calculation on different data.

This CPU-GPU complementarity fosters the implementation of tasks on the most suitable target, but it is not always the best

solution. Indeed, the data transfers between the two materials are very time-consuming[1] and must be prevented.

3.1. Partitioning methodology

Cuda is a high-level language[2] which allows the programmer to work whatever the GPU architecture, while it is an Nvidia graphic card. Nevertheless, some modifications' effects may depend on the architecture features or on the compiler used and cannot be determined a priori. This is why Nvidia provides a methodology[1] to optimize any cuda application. This methodology is an iterative optimization called APOD for its four steps: "Assess, Parallelize, Optimize, and Deploy.".

First, the bottleneck, the steps responsible for a large share of execution time, and the parallelizable tasks must be identified. Whenever possible, modifications should be made to the algorithm to improve the parallelism or make it more adapted for a GPU. Then the algorithm must be split up between the CPU and the GPU. When it is done, the code can be optimized to reduce the processing time. To do so, unit testing must be done to analyze the results and check the proper algorithm's functioning and consistency. Furthermore, time measurements to check the performance's effective increase are essential. Thus every software or algorithmic change has been integrated with a new function version. Then every function was compared to the other on random cases (with count agent's variation) and on a large and fixed case. Micro-benchmarks have been made to locate each kernel function on the roofline to compare more precisely when the processing times were too similar. The methodology for those tests is based on [29]. The time of multiple kernel functions' calls on random-generated data (copied between calls to prevent GPU optimization) is measured. Then, each kernel function's operational intensity and the operation count are analytically determined.

All algorithms, the study case and the micro-benchmarks can be found on the following Gitlab repertory[3].

[1] https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html
[2] https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[3] `https://gitlab.com/satie.sete/p2p_market_resolution_gpu`

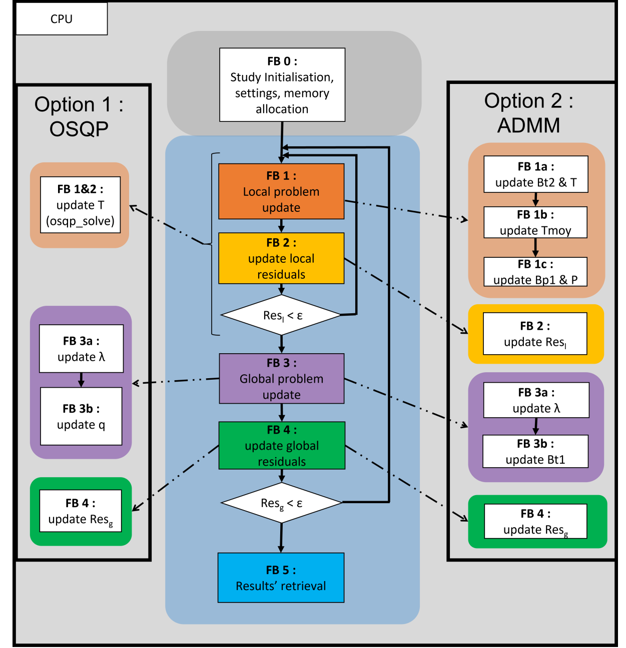

Figure 3: Algorithm's functional blocks, CPU version

3.2. CPU implementation

As explained above, the algorithm must be analyzed on the CPU before partitioning. As a reminder, the application principle is for one study case to compute the electricity consumption, production, and price. The input is a description of the study case with the agents' count (N), the set of each agent type (Ω_g,Ω_c,Ω_p), and for each the cost function (g_n) and the power bounds ($\underline{P_n}$, $\overline{P_n}$). This application can be used several times to change some parameters and simulate several time steps. The outputs are the negotiations' results, i.e. all trades (T) (and thus the total power for each agent P_n) and the price matrix Λ. The residuals and the complete step count k_g (the comprehensive communications count) represent the quality of the convergence. Finally, the computational time t_{simu} is measured for each application's use. The functional block representation of the CPU algorithm is made in Fig. 3; the description of the different blocs is as follows.

- **FB 0** is the data initialization corresponding to the study case and the algorithm choice. Then if ADMM is chosen, matrices creations (in-house class) are made. Otherwise, OSQP requires the creation of OSQP objects to store parameters and realize minimization.

- **FB 1** and **FB 2** are the local problem resolution. For the OSQP algorithm, this step uses the C OSQP solver once per agent. For the ADMM algorithm, this bloc is the repetition of the three operations (a, b, and c) to obtain the exchanges and the power to compute the residuals finally.

- **FB 3** is the global problem's update, i.e., the dual variable Λ update for the two algorithms, the b_{t1} update for ADMM, and the q update for OSQP.

- **FB 4** is the residual calculation for the two algorithms.

- **FB 5** is the retrieval of results in a class made for this.

The different steps of the resolution for the ADMM version are resumed in Tab. 3:

Table 3: Functional blocks and equations matching

Block	Description	equations
FB 0	Initialization	
FB 1a	Local problem (B_{t2}, T)	(11b) , (8a)
FB 1b	Local problem($\overline{T}$)	
FB 1c	Local problem (B_{p1}, $\tilde{P}$, μ)	(11c) (8b) (8c)
FB 2	Local residuals	(12) or (14)
FB 3a	Global problem (Λ)	(3)
FB 3b	Global problem (B_{t1})	(11a)
FB 4	Global residuals	(4) or (13)
FB 5	Results' retrieval	

The reader must notice that step **FB 3** will begin with the peer's communication in the real implementation. Nevertheless, in the simulation, all data is stored in the CPU and GPU global memory, which is accessible by all agents. To remain the closest possible to actual implementation, agents only read data they would have access to (*i.e.* their own data and the trade p_{mn} proposal of their peers m).

It is worth noticing that for the OSQP method, the local problem is solved agent per agent because clustering the agent increases the complexity. The OSQP objects for each agent are maintained between the steps to keep all parameters and results for the warm start and prevent many reboots. The linear cost vector (q in Appendix) is the only parameter that changes between steps. On the other hand, the ADMM processing is achieved by implementing matrices since this does not change the complexity, and that will be the same structure for the GPU version. Thus the result for three methods on CPU will be presented, **COSQP** is the Centralized OSQP, **OSQP** is the decentralized OSQP, and **ADMM** is the decentralized ADMM.

3.3. GPU implementation

This section will focus on the CPU-GPU partitioning, i.e., which blocks of the algorithm will be deported on the GPU. First, let's notice that according to [30], using OSQP on GPU[4] can only be helpful from a local problem size of $M_n = 10^5$. So it can be used for the centralized resolution as a reference. This new implementation called **COSQPGPU** is faster than the CPU centralized resolution (**COSQP**) but slower than all the decentralized methods. It will not be kept for the following sections. Furthermore, the decentralized method will not reach this number of agents and peers. By clustering the agents, the local problem's size can increase and make the GPU more interesting for OSQP. Nevertheless, clustering the agents will increase the complexity too much to be interesting so that the decentralized OSQP will be kept on the CPU.

In Fig. 4, the relative time for each functional block for one 220 agents study case on four consecutive market resolutions with Warm start for all methods can be seen. The GPU results will be discussed later in another section. The centralized method (**COSQP**) needs 20% of the computational time to initialize the study case; the OSQP solver uses the remaining time. For the decentralized CPU methods (**OSQP**, and **ADMM**), the more critical step is during the local problem resolution **FB 1**. Fortunately, for the ADMM, these calculations are linear operations adapted for a massively parallel architecture. Indeed agents read their own data and write their results, which are different for all agents in the global memory, so there is no race condition. Therefore, all ADMM calculations were implemented on GPU, and the CPU will be used to initialize data and manage the conditional loop of the algorithm. This will limit data transfers between the CPU and the GPU. Indeed, data

[4]https://github.com/ZhenshengLee/cuosqp

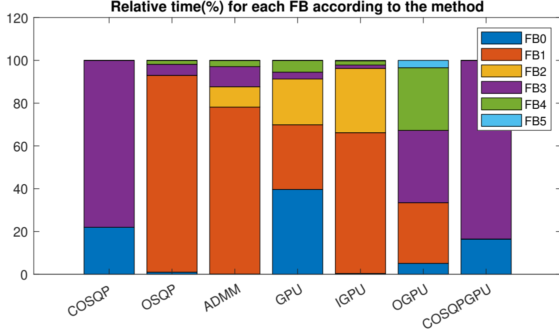

Figure 4: Relative time, $N = 220$ agents and four-step simulation

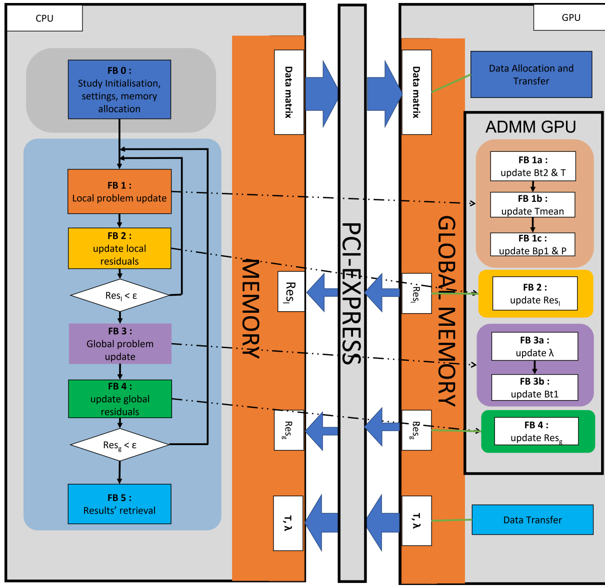

Figure 5: CPU-GPU partitioning of the algorithms functional blocks

available in the GPU memory can be reused by all the calculations on the GPU. The data transfers are only made at the algorithm's beginning and end (except for the residual, which is only a float variable). The resulting functional block partitioning is represented in Fig. 5. Every data transfer is depicted with blue arrows. It is worth noticing that if the first and last data transfer is $O(N^2)$ sized, the intermediate transfers are $O(1)$ sized (two floats every step). The dotted arrows represent the kernel calls.

Identification of the dependency between the data used for the calculation is necessary when the implementation is parallel. Each agent's peer results are needed for this algorithm's global update and residual calculation. Thus, this step implies a synchronization between the agents, *i.e.* that each agent must wait that all agents have finished their computation to continue running the algorithm. On the other hand, for each agent, all the other actions do not need the results of the other agents. Therefore, there is no need for synchronization between the agents for the local problem.

In this work, only the default stream of the GPU has been used. So only one kernel launch can be done at once. This means that there is a global synchronization between the kernel launch and at each data transfer, so the computation of the second kernel call begins only when the previous call is complete.

3.4. Algorithm rewriting

At least two ways of using OSQP are possible for the local problem. Either the trade substitutes the power p_n through the constraint (1c). Either the power p_n and the constraint (1c) are kept as a variable and constraint of the problem. This last choice slightly increases the size of the problem (one more constraint and variable) but makes the H matrix diagonal, which greatly increases the OSQP KKT matrix's sparsity.

As a GPU has more float calculation units than double-precision ones, using single-precision floats will increase the computation capability of the GPU. Nevertheless, it will impact the calculation accuracy and impose a minimum on the reachable residuals. To prevent any error accumulation on the residual calculation, this will be done by taking the maximum ($||\epsilon||_\infty$) rather than the sum of the coefficients ($||\epsilon||_2$). The residual calculations (4) become:

$$\begin{aligned} r^{k+1} &= \max_{n,m\in\Omega} (t_{nm}^{k+1} + t_{mn}^{k+1}) \\ s^{k+1} &= \max_{n,m\in\Omega} (t_{nm}^{k+1} - t_{nm}^{k}) \end{aligned} \tag{13}$$

The same goes for the local residual calculations (12) for each agent:

$$\begin{aligned} r_l^{j+1} &= \bar{t}^{j+1} - \tilde{p}^{j+1} \\ s_l^{j+1} &= \max_{i<M_n} (t_i^{j+1} - t_i^{j}) \end{aligned} \tag{14}$$

It is worth noticing that in a case where all agents are not linked together, many of the trade matrix terms are null. Computing these terms is useless and can slow down the simulation using

GPU resources. The problem can be rewritten to only consider non-null terms. The problem size remains N agents, but the computational and memory complexity reduce from $O(N^2)$ to $O(M)$ with ($M = \sum_{n\in\omega} M_n$) the total possible trades' count. To switch from one space to another, vectors have been introduced to store the correspondence for the index.

input : Study case, ρ, ϵ and kmax
output: Vectors T, Λ, residuals

```
1  while err > ε and k < kmax do
2      while err_l > ε and j < jmax do
3          B_t2 ← (11b) ;
4          T^{j+1} ← (8a);
5          T^{j+1} ← max(min(T^{j+1}, Ub), Lb) ;
6          T_mean ← mean(T^{j+1}) ;
7          B_p1 ← (11c) ;
8          P̃ ← (8b) ;
9          P̃ ← max(min(P̃, p̄_n/M_n), p_n/M_n) ;
10         μ ← (8c) ;
11         if j%Step_l == 0 then
12             (r_l^{k+1}, s_l^{k+1}) ← (14) ;
13             err_l ← max(r_l^{k+1}, s_l^{k+1}) ;
14         end
15     end
16     Λ ← (3) ;
17     B_t1 ← (11a) ;
18     if k%Step_g == 0 then
19         (r^{k+1}, s^{k+1}) ← (13) ;
20         err ← max(r^{k+1}, s^{k+1}) ;
21     end
22 end
```

Algorithm 3: Optimized algorithm on CPU-GPU

To limit the number of synchronizations, data transfer, and useless calculations, the residuals can be computed every $Step_l$ or $Step_g$ iterations (respectively for the local or global problem). If this number of steps between each computation is too small, there will be a lot of useless calculations of residual and data transfer; if it is too high, there will be a lot of useless iterations. The ADMM algorithm becomes Alg. 3, with T the M sized vector (rather than a $N \cdot N$ matrix).

In this work, every kernel call is blocking, as we use the default stream of the GPU. Thus, there is a synchronization between the kernel calls, which can be useless. Operations should be aggregated in the same kernel call to prevent useless synchronization. Nevertheless, all operations are not computed on the same-sized problem.

Indeed the **FB 3** and **1a** are calculations on M sized vectors, the **FB 1b** block is a reduction from M to N, and the **FB 1c** block is a calculation on N sized vector. Finally, the **FB 2** block is a N to 1 and a M to 1 reduction, and the **FB 4** bloc is two M to 1 reductions. Different sizes' calculations on the same kernel call will reduce the parallelism. Indeed, there will be many inactive threads, or each thread will do several computations. Furthermore, having only one thing by kernel call will facilitate the GPU compiler optimization and the branching prediction.

Due to its large number of processors (over a thousand), the GPU computation is particularly efficient in performing the same instruction on a set of data (SIMD: Single Instruction Multiple Data). Indeed, even if the GPU Streaming Multiprocessors (SM) can execute different instructions, only one instruction type can be computed on the processors in each SM. Thus, the local ADMM algorithm's parallelization, Alg. 2, must consider this constraint. The most natural thought would be to parallelize the agents by analogy with the real distributed deployment. The left part of Fig. 6 represents the agent parallelization for the functional blocks **FB1** and **FB2**. Nevertheless, this configuration raises some issues.

First, if the agents do not have the same peers count (for example, here in Fig. 6, 1 and 2 peers), the different branches will not compute the same number of operations. Furthermore, each agent finishes the calculation when it achieves its termination criterion. But, doing so provokes divergent embranchment between the threads (those who must continue the simulation and those who must finish). The next step (**FB 3**) is blocking; the agents that end will do nothing, waiting for the others. Thus, there will be many inactive threads, which will not save time. Indeed, there are two possibilities for an agent that ended: Either the agent thread is launched as part of an active warp and does nothing. This will anyway use some GPU resources. Either it is not launched, but in this case, contiguous threads may not access adjacent memories (no-coalescent memory access). On the other hand, the **FB1a** and **FB2** steps are not fully parallelized because, respectively, independent calculations are serialized, and the reduction is not parallelized.

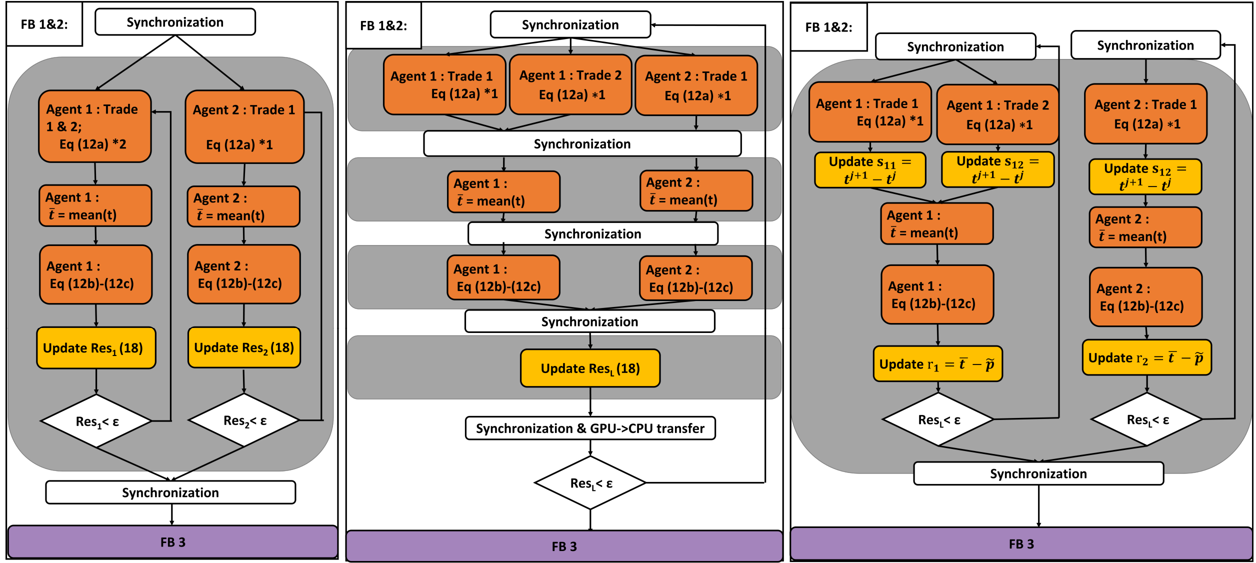

Figure 6: Agent parallelized with independent residual (left), trade parallelized (middle), and trade parallelized with independent residual (right), focus on two agents, one with two peers and the other with 1 peer. Grey boxes are kernel calls

In the middle of Fig. 6, the computation on trades is parallelized to prevent the size-varying for-loop, and the residual calculation is shared with every agent. All agents will stop when the maximum residuals are small enough. Each step is done by one kernel call (grey rectangle), and the size is always adapted to the computation, so the calculation is fully parallelized. Every thread processes precisely the same operation. Nevertheless, this is done by adding synchronization between each step (several kernel calls). The left part of fig. 6 shows the compromise between parallelization and using one kernel call. As in the first proposition, only one kernel call is used, and each agent has its residual. All are made by having one block of thread by agent and using shared memory to calculate one reduction by block. Thus, all steps are fully parallelized, but the **FB1c** only uses one thread by block (all the other threads must wait). One shared variable by block is used to specify if the block must continue to compute. This variable is set to false if one residual computation is higher than the precision asked and reset to true at each loop start. Several threads may want to write on this variable simultaneously, but there is no race condition as they all write the same value.

For this algorithm instance, the penalty factor ρ can significantly impact the convergence of the problem. If it is too large, the problem will not converge; if it is too small, it will converge in too many iterations. According to [26], the penalty factor can be changed according to the ratio between residuals to speed up the convergence. The demonstration of convergence also applies if the penalty factor becomes fixed after a finite number of iterations. All other modifications don't impact the convergence as it remains the exact computations. Using simple floating precision may prevent the algorithm from reaching very small residuals, but that precision is unnecessary in our problem.

3.5. *Software optimization*

This section focuses on the third part of the APOD methodology. Optimizations will not change the algorithm, only how it will be executed and will enhance the processing time.

First, these algorithms need fewer iterations to converge if they begin close to the optimum. Thus, the use of a warm start is possible, *i.e.* using the previously found optimum to initialize the computation as the consumption may not differ a lot between each time step. Furthermore, using objects and attributes saves memory allocation time (when the problem size remains

the same between two simulations) and limits data transfers between CPU and GPU when several simulations use the same data. Finally, the two coefficients B_{t2}, B_{p1} are computed but not stored to save two writings by step, and as A_{p2} is never used alone; thus, the new coefficient $A_{p12} = A_{p1} + A_{p2}$ is used to save one operation per iteration.

During the resolution, there are two different reductions. **FB 2** and **FB 4** are maximum searches, and **FB 1b** is a mean. The method proposed in [31] has been followed to optimize the reductions.

Finally, the last but not least step consists in tuning the compiler. Indeed several options can be used to improve the performance, such as using inline functions, disabling some unnecessary security options, and asking to optimize the speed of the application (rather than the storage space). Nevertheless, the black-box compiler can also provoke unexpected performance results after any optimization or algorithm re-writing. This is why evaluating the different implementations is very important.

3.6. Parallel methods' complexity

In this section, the Brent theorem [32] will be used to evaluate the complexity of the optimized algorithm on GPU. As a reminder, the problem has a size of $M = O(N^2)$.

Let o be the total calculations' count, and let t be the parallel complexity. **FB 3a&b** and **FB 1a** have a constant calculations' count per thread, thus $o_1 = O(M)$, $t_1 = O(1)$. With p_1 processors (or thread), the Brent theorem implies that the complexity becomes:

$$C_{para1} = O(\frac{o_1}{p_1} + t_1) = O(\frac{M}{p_1} + 1) \quad (15)$$

Thus, if $p_1 = M$, the calculation of those steps is constant time according to the problem size. The same goes for **FB 1c** with $o_1 = O(N)$.

FB 2 and **FB 4** are optimized reductions according to [31] on M-sized vector. The operation count is now $o_2 = O(M)$ and the parallel complexity is $t_2 = O(log(M))$. Similarly, **FB 2b** is an optimized reduction from an M-sized vector to an N-sized vector, so $o_2 = M = N^2$ and $t_2 = log(N) = log(M)/2$. The complexity is:

$$C_{para2} = O(\frac{o_2}{p_2} + t_2) = O(\frac{M}{p_2} + log(M)) \quad (16)$$

Thus having $p_2 = \frac{M}{log(M)}$ processors is enough to have a $O(log(M))$ complexity.

The best reachable complexity is $O(log(N))$ for this algorithm. Still, if the GPU cannot manage the processors' asked count (i.e., $p = p_{max}$), the complexity becomes $O(M) = O(N^2)$ as in the serialized algorithm. In this work, all kernel calls have $p = M$ or $p = N$ according to the calculation sized, except **FB 2b** where $p = blockSize * N$ because there must be one block per agent. Similarly, if the right part of Fig. 6 is made, $p = blockSize * N$ for all functional blocks of the local problem. It is why the complexity can be $O(log(M))$, $O(N)$, or $O(N^2)$ according to the problem size.

4. Results

4.1. CPU-GPU specifications

In this work, the implementation has been made on an Nvidia GPU GeForce RTX 3060. It is an Ampere architecture (GA 106) with 30 Streaming Multiprocessors (SM) and 3840 Cuda cores. According to the micro-benchmark of [33], the roofline of this GPU is given in Fig. 7. The roofline is an insightful model that can help programmers to characterize the hardware they use and determine the bottleneck of its algorithm (memory-bound or compute-bound) on this hardware. In this work, the operational intensity is low (about 1); thus, according to the roofline, the algorithm should be memory-bound. The measured performance can be better than the theoretical one because while the temperature is low, the GPU can increase its clock frequency higher than the theoretical value. The associated CPU is an AMD RYZEN 5 5600H operating at 3,3GHz.

The following part will present the results' comparison between the five implemented methods. There will be two methods on CPU called **ADMM** and **OSQP** and three methods on GPU called **GPU**, **IGPU** and **OGPU**. **GPU** is the first version

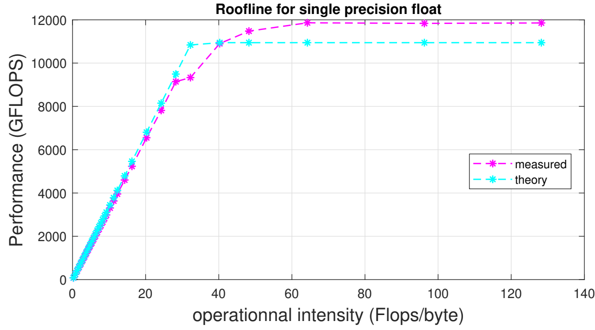

Figure 7: Single precision float roofline

of the algorithm on GPU, whereas **IGPU** is an optimized version following the principle of the middle of fig. 6. Finally, **OGPU** is the optimized version following the right part of fig. 6. The **GPU** method has one kernel call for each functional block (except for **FB 2b** and **FB 2c**, which are proceeded in the same call). The optimized versions (**IGPU** and **OGPU**) have, in addition, the following optimizations:

- every variable is stored with class attributes and kept on GPU between steps if possible, B_{t2}, B_{p1} are not stored;
- the penalty factor varies according to the residuals;
- the problem has been reduced from N^2 to M, A_{p12} is used;
- the T_{mean} computation is optimized according to [31].

All the other optimizations and algorithm re-writing viewed earlier are already used in the first GPU version and kept for optimized versions. Note that all optimizations that can also be applied on the CPU have been applied to the presented CPU method. The functional block study has also been made for the GPU methods and can be seen in Fig. 4. By comparing **GPU** and **IGPU**, it can be seen that keeping data on GPU reduces the relative time for the initialization **FB 0**. Comparing **IGPU** and **OGPU**, shows that having one kernel call for all the local problems **FB 1&2** reduces the associated time. Nevertheless, these kinds of measures can only be done by adding synchronization barriers that can greatly influence the algorithm performance; this is why further study without these barriers is necessary.

4.2. Problem size impact on the resolution time

Random cases have been created to evaluate the algorithm responsiveness to the size problem. These cases are made by generating the different agent features with a homogeneous random draw in a fixed interval. The consumer, generator, and prosumer proportions are set for these cases. The power bounds are unrelated to the cost function; thus, some agents can be very constrained in those cases. The study case parameters for the performance evaluations are given in Tab. 4, and the simulation parameters are shown in Tab. 5.

Table 4: Random characteristics for the study case generation

Features	average	variation
Power (MW)	1000	400
a (MW^{-2})	0.07	0.02
b (MW^{-1})	50	20
Consumer (%)	50	0
Prosumers (%)	12,5	0

Table 5: Parameters set for the simulation

Features	value	Features	value
k_{max}	10000	j_{max}	1000
$step_g$	5	$step_L$	5
ϵ_g	0.01	ϵ_l	0.005
ρ_g	$0.05 \cdot N$	ρ_l	ρ_g

Fig. 8 displays the average simulation time depending on the problem size and according to several methods. For each problem size, the simulation was done on 50 cases (the same for all methods), and the method's order was randomized for each case. For clarity, the average times according to the problem size are resumed in Tab. 6. It is worth noticing that GPU methods exhibit a strong acceleration in processing time, even at 100 agents.

Hence, even with a problem size increase, the simulation time of the methods on GPU remains minimal, and the differences between the GPU methods decrease. In contrast, the simulation time for the CPU methods (more than 10s) is prohibitive for a long-term study. To compare the ADMM algorithms with OSQP, the ratio between the OSQP time with their average sim-

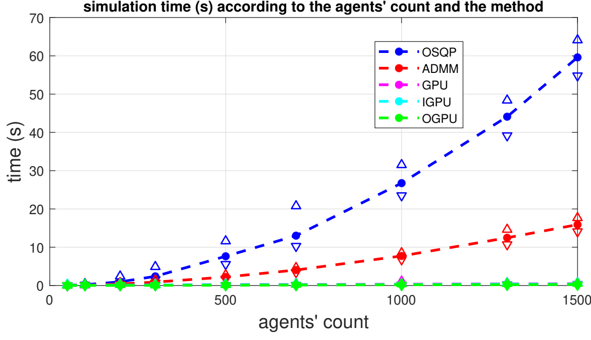

Figure 8: Minimum, maximum, and average simulation time for random cases

Table 6: Average times (s) according to the method and the problem size

N agents	100	500	1000	1500
OSQP	0.2	8	27	60
ADMM	0.1	2	8	16
GPU	0.05	0.2	0.4	0.4
IGPU	**0.07**	**0.2**	**0.3**	**0.4**
OGPU	**0.03**	**0.05**	**0.2**	**0.2**

ulation time has been written in Tab. 7. The CPU-GPU parti-

Table 7: Ratio between the OSQP and the other method average times

N agents	100	500	1000	1500
ADMM	2	3	3	4
GPU	5	32	75	**140**
IGPU	3	42	**105**	**167**
OGPU	8	**142**	**129**	**317**

tioning and the algorithm rewriting significantly impacted the simulation time. The resolution is made about 100 times faster with the GPU method than with the must-use method OSQP. To study the speed up of the CPU-GPU partitioning of the ADMM methods, it will be calculated for each study case s:

$$S_s = \frac{t_{s,ADMM} - t_{s,method}}{t_{s,ADMM}} \quad (17)$$

The average results are resumed in Tab. 8. Whatever the prob-

Table 8: SpeedUp in percentage for the ADMM CPU-GPU partitioning

N agents	100	500	1000	1500
GPU	54 %	**89 %**	**95 %**	**97 %**
IGPU	39 %	**92 %**	**97 %**	**98 %**
OGPU	**75 %**	**98 %**	**97 %**	**99 %**

lem size, the speedup is very high; for instance, the computation time is enhanced by more than 95% by using the GPU methods for a case of 1000 agents.

This type of study allows measuring the complexity and comparing it with theoretical results. Let η be the time augmentation due to the problem size increase for a given method. The expression is the following for the increase from N_1 to N_2 agents:

$$\eta = \frac{t_2}{t_1} \quad (18)$$

Let $O(N^{\alpha_{poly}})$ be the method complexity, it implies:

$$\eta = \frac{t_2}{t_1} = (\frac{N_2}{N_1})^{\alpha_{poly}} \quad (19)$$

Thus, the measured complexity can be calculated as follows:

$$\alpha_{poly} = \frac{log(\eta)}{log(N_2) - log(N_1)} \quad (20)$$

Tab. 9 aggregates the different simulation time increases for a rise from 50 to 1500 agents and the related complexity. The measure is coherent with the theory of the ADMM CPU method. The lower measured complexity of OSQP can be explained by the high sparsity of the OSQP problem matrices. The non-optimized GPU method computes the mean of the trade with a function that has a complexity of $O(N)$. This stays coherent with the value measured. For the optimized method, the logarithmic complexity ($O(log(N)^{\alpha_{log}})$) has also been computed with simulation time increases for a rise from 50 to 1500 agents and the related complexity

$$\alpha_{log} = \frac{\log(\eta)}{\log(\log(1500)/\log(50))} \quad (21)$$

It seems that the number of threads (here, 512 threads per block) was not enough to achieve a logarithmic complexity; it is why the complexity of the optimized GPU methods is not a logarithmic one.

4.3. Performance evaluation using a European market dataset

Thanks to the random cases, the complexity and the convergence's time have been evaluated. The GPU methods show a

Table 9: time increase, measured complexity and theoretical one (with K a constant depending on the block size)

Method	η	α_{poly}	α_{log}	C_{the}
OSQP	1216	2.1		$O(N^3)$
ADMM	656	1.9		$O(N^2)$
GPU	15	0.8	4.3	$O(N)$
IGPU	5.5	0.6	2.5	$O(\frac{N}{K} + log(N))$
OGPU	47	1.1	6.4	$O(\frac{N}{K} + log(N))$

remarkable improvement in these factors compared to the CPU ones. Nevertheless, these random cases are not related to real study cases. Thus, this section will focus on a real study case: the European market. This case uses open access data DTU-ELMA/European Dataset [34]. This dataset includes a generator file with their names, locations, types (coal, lignite, hydraulic, gas, nuclear, geothermal), capacities, and production costs. It also includes nodes description (voltage, location, and load) at each instant with one hour step for three years. These data allow simulation of a long period of the market with coherent consumption data.

This dataset includes 969 fully controllable generators and 1494 nodes considered consumers. Their capacity bounds the generator power; the measured load on the data within 10% determines the consumer power. The function cost of the consumers has its minimum at the measured load and a quadratic term at 1. The generator function cost has its linear term equal to the linear cost of the production and a quadratic coefficient at 0,1. Thus it is more important to satisfy the consumers than the generators. As a reminder, the cost function is $g(P_n) = a \cdot P_n^2 + b \cdot P_n$, the coefficient choices are in Tab. 10, and the simulation parameters are the same as those in Tab. 5.

To evaluate, compare and present the performance of these methods, the date of January 2, 2013, was arbitrarily chosen.

Table 10: Coefficients' definition

Agent	Producer	Consumer
$\underline{p_n}$ (MW)	0	$-1.1 \cdot P0$
$\overline{p_n}$ (MW)	capacity	$-0.9 \cdot P0$
a (MW^{-2})	0.1	1
b (MW^{-1})	production cost	P0

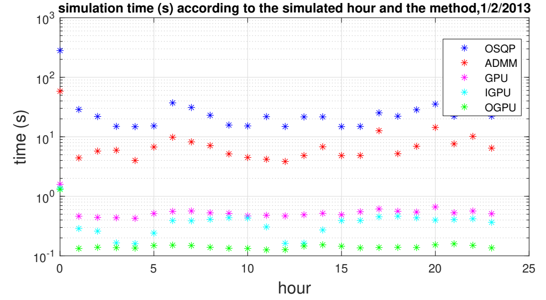

Figure 9: Computation time according to the simulated hour

The results are similar, whatever the selected date. Consecutive time steps have been chosen to use the warm start; indeed, the first hour is way much slower than it could have been with a warm start (by beginning one hour earlier).

It is important to note on Fig. 9 that the computation time fluctuates slightly according to the consumer demands or their variations (which are the two things that differ between two hours). But whatever the considerate hour, GPU methods are faster than CPU ones, and ADMM is better than OSQP. For clarity, results have been summarized in Tab. 11.

Table 11: Average, median, minimum, and maximum computation time in seconds according to the implemented method

Method	Mean	Med	Min	Max	Std
OSQP	33	22	15	281	53
ADMM	8.9	6.2	3.9	58	11
GPU	0.56	0.52	0.42	1.5	0.2
IGPU	**0.38**	**0.39**	**0.16**	**1.4**	**0.2**
OGPU	**0.19**	**0.13**	**0.13**	**1.3**	**0.23**

Furthermore, the optimized method solves a 2463 agents' study case in less than 0.2 seconds. This means that one year can be simulated in less than half an hour. In Tab. 12, can be found the average, minimum, and maximum absolute Speed up (17) for the ADMM methods. As previously, the speedup is calculated for each study case (here for each time step), then then the minimum, the maximum, and the mean of the results are taken.

It can clearly be noted that the optimized method is faster than the others. This acceleration is pretty constant in this study, whatever the consumers' needs. This speed-up makes possible simulations on large-scale cases and periods.

Table 12: Absolute Speedup of the GPU methods compared to the CPU ADMM method

Method	Mean	Med	Min	Max	Std
GPU	92 %	92 %	88 %	97.1 %	0.025
IGPU	**95 %**	**95 %**	**90 %**	**98 %**	0.021
OGPU	**98 %**	**98 %**	**97 %**	**99 %**	0.0065

5. Conclusion

In summary, the significant contributions of this work are the following:

- a comparison of the computational and memory complexity of several algorithms for P2P market resolution;
- a computational complexity study of the ADMM algorithm compared to state-of-the-art methods
- rewriting the ADMM method for a CPU-GPU partitioning and a parallel implementation
- performance and complexity evaluations on random cases, a European dataset, and a comparison to state-of-the-art methods

The results show the speedup achieved with the algorithmic rewriting, the CPU-GPU parallelization, and software optimization as part of a hardware-software codesign approach. The European case was used to show a more realistic case with coherent time variation. The resolution on GPU requiring less than 0,2s (against several hours with MATLAB and several seconds with C++ on CPU) per hour allows the simulation of one year in less than half an hour for a system of 2463 agents. All algorithms, the study case, and the microbenchmarks can be found on the following Gitlab repertory[5]. Nevertheless, a more significant acceleration may be reachable. First, the peers' agents count can be limited, reducing the complexity from $O(N^2)$ to $O(N)$. But studies must be done to select the best peers possible for each agent. Furthermore, a decreasing local termination criterion may speed up the beginning of the simulation where a precise solution is not needed for the local problem. Finally, some constant parameters can be stored in the constant memory of the GPU to speed up the memory access.

Similarly, this work has shown one CPU-GPU partitioning to speed up the resolution of this problem. The codesign approach is not limited to CPUs and GPUs: FPGAs would also be of interest. Their massively parallel architecture would bring about a specific adequacy and perhaps allow new computing time improvements. Moreover, the complementarities between CPU, GPU and FPGA would open the discussion of another metric to quantify the performance of a resolution: its energy cost when scaling the problem size

[5] https://gitlab.com/satie.sete/p2p_market_resolution_gpu

6. Acknowledgments

The authors want to thank the Cairn team (Energy-Efficient Computing Architectures) of the IRISA laboratory of Rennes composed by Mickaël DARDAILLON, Simon ROKICKI and Steven DERRIEN for their expert advice on this work.

References

[1] RTE. Conditions and Requirements for the Technical Feasibility of a Power System with a High Share of Renewables in France Towards 2050

[2] Bussar C, Stöcker P, Cai Z, Moraes Jr. L, Magnor D, Wiernes P, Van Bracht N, Moser A, Uwe Sauer D. Large-scale integration of renewable energies and impact on storage demand in a European renewable power system of 2050—Sensitivity study, Journal of Energy Storage, Volume 6, 2016, Pages 1-10, ISSN 2352-152X, https://doi.org/10.1016/j.est.2016.02.004.

[3] Saad Al-Sumaiti A, Ahmed M H, Salama M M. (2014) Smart Home Activities: A Literature Review, Electric Power Components and Systems, 42:3-4, 294-305, doi: https://doi.org/10.1080/15325008.2013.832439

[4] Sousa T, Soares T, Pinson P, Moret F, Baroche T, Sorin E. Peer-to-peer and community-based markets: A comprehensive review, Renewable and Sustainable Energy Reviews, Volume 104, 2019, Pages 367-378,ISSN 1364-0321 https://www.sciencedirect.com/science/article/pii/S1364032119300462

[5] Dong A, Baroche T, Le Goff Latimier R, Ben Ahmed, H. Convergence analysis of an asynchronous peer-to-peer market with communication delays. Sustainable Energy, Grids and Networks, 26, 100475. 2021

[6] Garau M, Ghiani E, Celli G, Pilo F, Corti S. Co-simulation of smart distribution network fault management and reconfiguration with LTE communication. Energies, 2018, 11.

[7] Baroche, T., Moret, F., Pinson, P. Prosumer markets: A unified formulation. PowerTech (pp. 1-6). IEEE. 2019

[8] Tushar W, Yuen C, Saha T K, Morstyn T, Chapman A C, Alam M J E, Poor H V . Peer-to-peer energy systems for connected communities: A review of recent advances and emerging challenges. Applied Energy. 2021

[9] Tushar W, Saha T K, Yuen C, Smith D, Poor H V. Peer-to-peer trading in electricity networks: An overview. IEEE Transactions on Smart Grid, 11(4), 3185-3200. 2020

[10] Ryan H, Marqusee J. "Designing resilient decentralized energy systems: The importance of modeling extreme events and long-duration power outages." Iscience (2021): 103630.

[11] Zhang X, Flueck A J, Nguyen C P. Agent-based distributed volt/var control with distributed power flow solver in smart grid. IEEE Transactions on Smart Grid, 7(2), 600-607.2015

[12] X. Su, C. He, T. Liu and L. Wu, "Full Parallel Power Flow Solution: A GPU-CPU-Based Vectorization Parallelization and Sparse Techniques for Newton–Raphson Implementation," in IEEE Transactions on Smart Grid, vol. 11, no. 3, pp. 1833-1844, May 2020, doi: 10.1109/TSG.2019.2943746

[13] Araújo I, Tadaiesky V, Cardoso D, Fukuyama Y, Santana Á . Simultaneous parallel power flow calculations using hybrid CPU-GPU approach. International Journal of Electrical Power & Energy Systems. 2019 .

[14] Sooknanan D J, Joshi A . GPU computing using CUDA in the deployment of smart grids. In 2016 SAI Computing Conference (SAI) (pp. 1260-1266). IEEE.2016, July

[15] Kargarian A, Mohammadi J, Guo J, Chakrabarti S, Barati M, Hug G, Baldick R. Toward distributed/decentralized DC optimal power flow implementation in future electric power systems. IEEE Transactions on Smart Grid, 9(4), 2574-2594. 2016

[16] Roberge V, Tarbouchi M, Okou F. Optimal power flow based on parallel metaheuristics for graphics processing units. Electric Power Systems Research, Volume 140, 2016, Pages 344-353, ISSN 0378-7796,

[17] Chernova T, Gryazina E. Peer-to-peer market with network constraints, user preferences and network charges. International Journal of Electrical Power & Energy Systems,Volume 131,2021,106981,ISSN 0142-0615, https://doi.org/10.1016/j.ijepes.2021.106981.

[18] Baroche T, Le Goff Latimier R, Pinson P, Ben Ahmed H. Exogenous Cost Allocation in Peer-to-Peer Electricity Markets. IEEE Transactions on Power Systems, Institute of Electrical and Electronics Engineers, 2019, 34 (4), pp.2553 - 2564. ffhal-01964190f

[19] Sorin E, Bobo L, Pinson P . Consensus-based approach to peer-to-peer electricity markets with product differentiation. IEEE Transactions on Power Systems, 34(2), 994-1004. 2018

[20] Santos G, Pinto T, Praça I, Vale Z. MASCEM: Optimizing the performance of a multi-agent system. Energy. 111. 513-524. 2016. doi: https://doi.org/10.1016/j.energy.2016.05.127.

[21] Martelli M, Enderli C, Gac N, Vermesse A, Merigot A. GPU Acceleration: OpenACC for Radar Processing Simulation. 2019 International Radar Conference (RADAR), 2019, pp. 1-6, doi: https://doi.org/10.1109/RADAR41533.2019.171296

[22] Balla-Arabe S, Gao X, Ginhac D, Yang F.Shape-constrained level set segmentation for hybrid CPU–GPU computers. Neurocomputing. Volume 177. 2016. Pages 40-48. ISSN 0925-2312. https://doi.org/10.1016/j.neucom.2015.11.004.

[23] Dine A, El Ouardi A, Vincke B, Bouaziz S, Speeding up graph-based SLAM algorithm: A GPU-based heterogeneous architecture study. 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP). 2015. pp 72-73, doi: https://doi.org/10.1109/ASAP.2015.7245711

[24] Pinson P, Baroche T, Moret F, Sousa T, Sorin E, You S. The emergence of consumer-centric electricity markets. Distribution & Utilization, 34(12), 27-31. 2017

[25] Stellato B, Banjac G, Goulart P, Bemporad A, Boyd, S. OSQP: an operator splitting solver for quadratic programs. Mathematical Programming Computation https://doi.org/10.1007/s12532-020-00179-2

[26] Boyd S, Parikh N, Chu Borja Peleato E, Eckstein J, Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers

[27] G Hug, S Kar, C Wu, "Consensus + Innovations Approach for Distributed Multiagent Coordination in a Microgrid," in IEEE Transactions on Smart Grid, vol. 6, no. 4, pp. 1893-1903, July 2015, doi: https://doi.org/10.1109/TSG.2015.2409053.

[28] Li X, Li F. GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method. Electric Power Systems Research. Volume 116. 2014. Pages 87-93 .ISSN 0378-7796, https://doi.org/10.1016/j.epsr.2014.05.005.

[29] Ofenbeck G, Steinmann R, Caparros V, Spampinato D G, Püschel M. Applying the roofline model. 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, pp. 76-85, doi: https://doi.org/10.1109/ISPASS.2014.6844463

[30] Schubiger M, Banjac G, and Lygeros J. GPU Acceleration of ADMM for Large-Scale Quadratic Programming

[31] Harris M. NVIDIA Developer Technology. Optimizing Parallel Reduction in CUDA

[32] Brent R. P. 1974. The Parallel Evaluation of General Arithmetic Expressions. J. ACM 21, 2 (April 1974), 201–206. https://doi.org/10.1145/321812.321815

[33] Konstantinidis E, Cotronis Y. A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling. Journal of Parallel and Distrind Computing. 2017. vol 107. p 37-56.

[34] Jensen T, Pinson P. RE-Europe, a large-scale dataset for modeling a highly renewable European electricity system. Sci Data 4, 170175 .2017. https://doi.org/10.1038/sdata.2017.175

[35] Davis T A. Algorithm 849: A concise sparse Cholesky factorization package. ACM Transactions on Mathematical Software (TOMS), 31(4), 587-591.(2005).

Consensus and sharing ADMM

This section will focus on giving more details on the equation demonstrations. Let the objective function be:

$$\underset{T}{\text{argmin}} \sum_{n}^{N} f_n(T) \tag{.1}$$

The global consensus problem for the ADMM [26] allow us to rewrite this minimization by:

$$\underset{T}{\text{argmin}} \sum_{n}^{N} f_n(T_n) \tag{.2a}$$

$$s.t\ T_n - z = 0 \tag{.2b}$$

with $f_n(T_n) = g_n(p_n) + \sum_{m\in\omega_n} \gamma_{nm} t_{nm}$ and the so-called common global variable:

$$z_{nm} = \frac{t_{nm} - t_{mn}}{2} \tag{.3}$$

and the augmented Lagrangien:

$$\begin{aligned} L_\rho(T, P, \Lambda, z) = \sum_{n\in\Omega} g_n(p_n) + \sum_{n\in\Omega}\sum_{m\in\omega_n} \gamma_{nm} t_{nm} + \\ \sum_{n\in\Omega}\sum_{m\in\omega_n} \left(\lambda_{nm}(t_{nm} - z_{nm}) + \frac{\rho}{2}(t_{nm} - z_{nm})^2\right) \end{aligned} \tag{.4}$$

The resulting ADMM algorithm is the following:

$$T_n^{k+1} = \underset{T_n}{\text{argmin}}\, f_n(T_n) + \sum_{m\in\omega_n} \lambda_{nm}^k (t_{nm} - z_{nm}^k) + \frac{\rho}{2}\|t_{nm} - z_{nm}^k\|_2^2 \tag{.5a}$$

$$\lambda_{nm}^{k+1} = \lambda_{nm}^k + \frac{\rho}{2}(t_{nm}^{k+1} + t_{mn}^{k+1}) \tag{.5b}$$

The sharing problem has the following form:

$$\begin{aligned} T_n^{k+1} = \underset{t_{nm}}{\text{argmin}} \quad & g_n\Big(\sum_{m\in\omega_n} t_{nm}\Big) + \sum_{m\in\omega_n} f_{nm}(t_{nm}) \\ \text{subject to} \quad & t_{nm} \in C \end{aligned} \tag{.6}$$

with:

$$f_{nm}(t_{nm}) = \gamma_{nm} t_{nm} + \frac{\rho}{2}\Big(t_{nm} - \frac{t_{nm}^k - t_{mn}^k}{2} + \frac{\lambda^k}{\rho}\Big)^2 \tag{.7}$$

The final algorithm became:

$$t_i^{j+1} = \underset{lb_n \le t_i \le ub_n}{\text{argmin}} \left(f_i(t_i) + \frac{\rho_l}{2}\|t_i - t_i^j + \bar{t}^j - \tilde{p}^j + \mu^j\|_2^2\right) \tag{.8a}$$

$$\tilde{p}^{j+1} = \underset{\underline{p_n} \le M_n \tilde{p} \le \overline{p_n}}{\text{argmin}} \left(g(M_n\tilde{p}) + \frac{M_n\rho_l}{2}\|\tilde{p} - \mu^j - \bar{t}^{j+1}\|_2^2\right) \tag{.8b}$$

$$\mu^{j+1} = \mu^j + \bar{t}^{j+1} - \tilde{p}^{j+1} \tag{.8c}$$

Complexity of the OSQP algorithm

A well-known state-of-the-art method when it comes to solving quadratic problems is OSQP[6] [25]. It will be considered in this work as a reference to evaluate the resolution complexity of a quadratic problem defined as follows:

$$\begin{aligned} x = \text{argmin}\ & \frac{1}{2}x^T H x + q^T x \\ \text{s.t.}\quad & l \le Ax \le u \end{aligned} \tag{.1}$$

Let N_c be the number of constraints (number of lines in A) and N_v the number of variables (length of x). OSQP is an iterative algorithm whose principle can be summed up as Alg.4. This algorithm's most important and costly step is a system solving, (.2), which requires a matrix inversion. Other update steps have complexity in $O(N_c)$ or $O(N_v)$.

$$\begin{pmatrix} H + \sigma I_{M_n} & A^T \\ A & -\rho_l^{-1} I_{M_n+1} \end{pmatrix} \begin{pmatrix} \tilde{x}^{j+1} \\ v^{j+1} \end{pmatrix} = \begin{pmatrix} \sigma x^j - q \\ z^j - \rho_l^{-1} y^j \end{pmatrix} \tag{.2}$$

input : j_{max}, ϵ, P, q, A, b, x0
output: x_{min}
1 $\tilde{z}, \tilde{x}, x, z$ *init*;
2 **while** $err > \epsilon$ *and* $j < jmax$ **do**
3 $(\tilde{x}^{j+1} v^{j+1}) \leftarrow$ solve system (.2) ;
4 $\tilde{z}^{j+1}, x^{j+1}, z^{j+1}, y^{j+1} \leftarrow$ linear update ;
5 $err \leftarrow$ linear update ;
6 **end**

Algorithm 4: OSQP algorithm

The higher dimension being the KKT matrix, the memory complexity of OSQP is $O((N_v + N_c)^2)$. Nevertheless, since OSQP only store sparse matrix under CSC format, this complexity may be reduced according to the problem at $O(N_v +$

[6] `https://github.com/oxfordcontrol/osqp`

$N_c) \leq O(H_{nz} + A_{nz}) \leq O((N_v + N_c)^2)$ with X_{nz} the number of non-zeros element in the X matrix.

Regardless of the method used for the system solving (conjugate gradient or LDL factorization), the complexity will not be linear, but polynomial $O((N_v + N_c)^\alpha)$ with $\alpha \geq 3$ for dense matrix. Once the factorization is done, solving the system has a complexity of about $O((N_v + N_c)^2)$. If the KKT matrix does not change between each iteration, its factorization can be done only once. In our case, only ρ_l can change depending on the OSQP settings but not at every step. Let K be the operation count where the factorization can be kept. The complexity can be written as $O((N_v + N_c)^\alpha + K \cdot (N_v + N_c)^2)$ which asymptotically is $O((N_v + N_c)^\alpha)$. However, the $O(K \cdot (N_v + N_c)^2)$ can be dominant for a small number of agents as K can happen to be larger than 1000 depending on the study case. With sparse matrix, the complexity can be smaller [35].