A Comparison of Three Parallel Processing Methods for a Resource Allocation Problem in the Smart Grid

B. Celik*, S. Suryanarayanan†, A. A. Maciejewski†, H. J. Siegel†, S. Sharma†, and R. Roche*
*FEMTO-ST, CNRS, Univ. Bourgogne Franche-Comté, UTBM, 90010 Belfort Cedex, France
Emails: {berk.celik; robin.roche}@utbm.fr
†Dept. of Electrical & Computer Engineering, Colorado State University, Fort Collins, Colorado, USA, 80523
Emails: {sid; aam; hj; swagata.sharma}@colostate.edu

***Abstract*—Aggregators are market participants that bridge the gap between the bulk electricity market and the emerging active end-user (smart home) by efficiently scheduling or allocating resources to meet certain objectives in the electricity grid. The computational burden and processing time of such allocation problems increases with the number of resources. Using high performance computing and parallel processing techniques, the computational time for simulating the environment can be managed. In this paper, we present and compare three parallel processing techniques executed on a dedicated high performance computer for simulating a multi-day aggregator-based resource allocation problem in the Smart Grid.**

I. Introduction

Applications of demand response (DR) are starting to play vital roles in the operation of smart electric distribution systems. The Federal Energy Regulatory Commission (FERC) defines DR as the set of "changes in electricity usage by end-use customers from their normal consumption patterns in response to changes in the price of electricity, or incentive payments designed to induce lower electricity use at times of high wholesale market prices or when system reliability is jeopardized" [1]. Accordingly, DR enables the participation of end-users in the electricity market by providing some opportunity of control of residential assets, behind the meter of a smart home, in return for a financial profit (or savings in expenditure) [2], [3].

As the number of smart homes participating in such DR actions increases, the need for a market entity to bridge the gap between the electricity market and the end-users is imperative. An aggregator serves the purpose of the third party market entity and reduces the operation burden of the utility and provides the customers with low cost consumption opportunities. The objective of the aggregator is to sell DR services to the operator and reduce the operation cost while providing benefits of savings to the end-users [4]. To achieve that objective, large and highly complex optimization problems must be solved by the aggregator daily. Thus, any new control technique proposed for aggregation should be tested by extensive simulations for long periods of time (i.e., days, weeks, etc.) before implementation.

In that regard, the performance of optimization techniques and the computation burden of the optimization algorithms require detailed study. Different optimization techniques for DR, including linear programming [5], mixed integer programming [6], particle swarm optimization [7], and genetic algorithms [3], [8] have been studied [9]. Although heuristic methods are applied to these complex problems to find near optimal solutions in polynomial time, computation times may increase rapidly as the size of the problem grows. Therefore, using parallel processing with multi-core computer in a high performance computing (HPC) environment is essential for reducing simulation time [10].

Open Multi-Processing (OpenMP) and Message Passing Interface (MPI) are the most commonly used programming styles in various applications for parallel processing [11]. The basic difference between these two styles is that the former uses a shared-memory architecture and the latter uses a distributed-memory architecture. Although OpenMP and MPI have different qualities, both are effective and efficient for achieving reductions in computation/simulation time. Further, a hybrid model incorporating OpenMP and MPI can be applied if the optimization problem and the computer structure are suitable to implement the features of these approaches. Programmers may achieve better results with the hybrid model.

In this paper, we present and compare three programming approaches—namely, OpenMP, MPI, and hybrid—for applying parallel processing methods to a resource allocation problem (such as customer appliance scheduling) involving aggregators for DR. It is not our intent to generalize the advantages and misgivings of these methods; rather, we aim to describe how these three approaches can be used in this resource allocation problem to help programmers and to compare their effectiveness vis-à-vis the reduction in simulation

This work was partly funded by the NSF Awards ECCS-1608898, and the Univ. of Bourgogne Franche-Comté, France. This work utilized the RMACC Summit supercomputer, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University. The RMACC Summit supercomputer is a joint effort of the University of Colorado Boulder and Colorado State University.

time.

The rest of the paper is organized as follows: Section II introduces the resource allocation problem; Section III details the OpenMP, MPI, and hybrid methods and their application to the resource allocation problem; Section IV presents the results and Section V concludes the paper.

II. SGRA Problem

A. Overview

In this paper, we use the aggregator-based residential DR program, denoted smart grid resource allocation (SGRA), from [12] as the test problem to implement our parallelization methods. For completeness, the SGRA problem statement is presented verbatim here: "given a set of customers and information about their respective assets, subject to customer constraints (i.e., availability of customer assets and customer incentive requirements), how can the aggregator find an incentive pricing and schedule of assets to maximize aggregator profit?" [12]. Note that we use the same DR program in [12] for multiple days to demonstrate the implementation of the above-mentioned parallelization techniques.

B. System Model

We consider an electricity network formed by $\mathcal{I}$ smart homes and one residential aggregator that interacts with the customers, the distribution utility, and the bulk electricity spot-market. In smart homes, electric appliances are divided into two groups: non-controllable (base) and controllable (assets). Totally, 31 types of base load and 18 types of asset are used to model the consumption profile of the smart homes. Appliance ownership is determined once, and the operation parameters are generated probabilistically for each day considered in the study.

Here, a day-ahead centralized control methodology is applied, wherein the aggregator controls the assets in the smart homes. The number of assets in each smart home is denoted by $\mathcal{A}_i$. End-users only decide the scheduling window of the assets; the aggregator determines the optimum start time of each asset. To convince customers to participate in the transactions with the aggregator, a customer incentive pricing (CIP) is offered by the aggregator [12]. The CIP is competitive compared to the utility's price for residential customers. With CIP, customers are expected to be more willing to allow scheduling of their assets and purchase electric energy from the aggregator rather than the utility. Thus, the aggregator aims to maximize its own profit as well as reduce the daily electricity cost incurred by the customer. As in [12], we use the PJM spot market [13] and the ComEd residential real-time pricing (RRTP) [14] for modeling the bulk electricity and the utility prices, respectively.

C. Problem Formulation

Prior to each day, the aggregator receives information on the utility and the spot market prices and the scheduling interval of assets from smart homes. After that, the aggregator determines the operation start time of assets with the CIP. The constraint for the scheduling interval of assets is formulated by (1).

$$\left[t^{s}_{a,i}, t^{e}_{a,i}\right] \subseteq \left[t^{s\_sch}_{a,i}, t^{e\_sch}_{a,i}\right] \tag{1}$$

where, $t^{s}_{a,i}$ and $t^{e}_{a,i}$ are the operation start and end times of the assets, respectively, and $t^{s\_sch}_{a,i}$ and $t^{e\_sch}_{a,i}$ are the user-defined acceptable operation start and end times of assets, respectively. However, customers may not be willing to allow rescheduling of their assets in return for modest savings; rather, they might expect significant savings for their efforts in participating in the DR program. To consider that, we use the same model from [12] to determine a threshold profit for each asset. The constraint for the threshold profit to allow rescheduling of an asset is given by (2).

$$\gamma_{a,i} = \begin{cases} 1, & \textbf{if} \quad c^{sch}_{a,i} \leq \alpha_{a,i} \cdot c^{0}_{a,i} \\ 0, & \textbf{otherwise} \end{cases} \tag{2}$$

where, $\gamma_{a,i}$ is the customer willingness for allowing rescheduling of an asset ($\gamma_{a,i} = 0$ disallows and $\gamma_{a,i} = 1$ allows rescheduling); $\alpha_{a,i}$ is a threshold metric in percent; $c^{sch}_{a,i}$ and $c^{0}_{a,i}$ are the costs of consumption with and without rescheduling, respectively. The costs for rescheduled and non-rescheduled conditions are calculated by (3) and (4), respectively.

$$c^{sch}_{a,i} = \sum_{t=t^{resch}_{a,i}}^{t=t^{resch}_{a,i}+d_{a,i}-1} \lambda(t) \cdot p_{a,i} \cdot \triangle t \tag{3}$$

$$c^{0}_{a,i} = \sum_{t=t^{s}_{a,i}}^{t=t^{e}_{a,i}} r(t) \cdot p_{a,i} \cdot \triangle t \tag{4}$$

where, $\lambda(t)$, $r(t)$, and $p_{a,i}$ are the CIP, the utility price, and the power rating of an appliance, respectively; $t^{resch}_{a,i}$, $d_{a,i}$, and $\triangle t$ are the determined asset start time, duration of asset operation, and simulation time interval (in this case, it is 0.25 representing 15 minutes). According to (2), the aggregator can only reschedule an asset if the cost reduction satisfies the threshold condition. Otherwise, the aggregator is disallowed from controlling the asset. Lastly, the optimization problem for maximizing the profit of the aggregator is given by (5):

$$\begin{gathered} \text{maximize} \left[P = \sum_{i=1}^{\mathcal{I}} \sum_{a=1}^{\mathcal{A}_i} \gamma_{a,i} \cdot (N_{a,i} + S_{a,i} - B_{a,i}) \right] \\ \text{subject to (1), (2)} \end{gathered} \tag{5}$$

where P is the profit of the aggregator; $S_{a,i}$ is the income from selling energy to a customer; $N_{a,i}$ is the income from selling negative load (i.e., the deferred peak) to the spot market; and $B_{a,i}$ is the expense for buying energy from the spot market. The detailed calculations of $S_{a,i}$, $N_{a,i}$, and $B_{a,i}$ are given in [12].

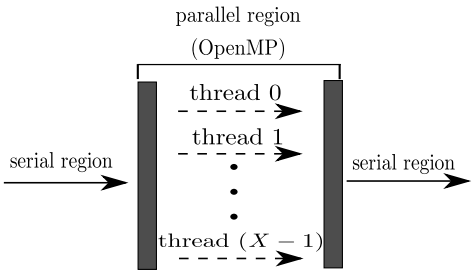

Fig. 1. OpenMP flow model.

III. Parallel Processing

A. Overview

Parallel processing is used for dividing a large problem into smaller sub-problems to solve them simultaneously using multi-core computer. In this problem, however, it is noted that the result of a sub-problem must not affect the result of any other sub-problem; thus, each sub-problem must be modeled independently.

In this section, we introduce the basics of parallel programming using the OpenMP and MPI programming approaches. Further, we use the above-mentioned approaches in three parallelization schemes for solving a multi-day SGRA problem.

B. Parallelization with OpenMP Programming

OpenMP is an application programming interface that provides a parallel processing framework using multi-threading (X threads $\rightarrow$ X cores) on a shared-memory architecture [15]. The set of threads/cores run simultaneously, i.e., in parallel, to execute sub-tasks or solve sub-problems. It is important to note that the specified task is divided among the threads and each thread has access to the same information (e.g., variable, parameter, objects) in the shared-memory. OpenMP supports multi-processing programming in C, C++ and Fortran languages on most platforms [16]. The OpenMP flow model is given in Fig. 1.

In the SGRA study, to solve the optimization problem with the OpenMP parallelization, the smart appliances are distributed among X number of threads to determine the profit of the aggregator. Fig. 2 depicts an implementation of the OpenMP programming for SGRA.

In the shared-memory, each thread (associated with a computer core) accesses a certain number of assets and individually determines $N_{a,i}$, $S_{a,i}$, and $B_{a,i}$ from (5) for each asset. After that, the aggregator profit, P, is determined for every asset rescheduled and then each thread sums the determined P values with every other thread to create one cumulative result for the aggregator profit.

C. Parallelization with MPI Programming

MPI is a specification for message-passing library interface that addresses parallel programming models on a distributed-memory architecture [17]. MPI is a communication protocol that supports both point-to-point and collective communication routines and was developed for cooperative parallel computing among computer cores running on distributed memory. With MPI, multiple tasks run simultaneously on separate cores defined by the user and each core has its own private memory. MPI programming does not use a shared-memory architecture; thus, cores are not able to access the same information stored in the memory. Therefore, cores need to use a messaging protocol, standardized by the MPI, if information from other cores are needed. Language bindings of MPI are defined for C, C++ and Fortran. Fig. 3 depicts the MPI application model. The basic difference between OpenMP and MPI is the written code is run for defined number of MPI task times simultaneously while OpenMP is solving optimization problem in parallel according to user-defined method (in this case, assets parallelization) inside the task.

To implement the parallel processing of SGRA using MPI, the total number of days (in the multi-day SGRA problem) is distributed among cores. The goal of this effort is to determine the aggregator profit for each day using a single core. For each day, the optimization problem (representing the same application albeit with different daily input) is solved simultaneously by cores in a distributed memory for each task. Message passing is not needed in this implementation because each day's optimization is totally independent from the other days; hence, no core has the need to wait for a message from another. The implementation of the MPI parallelization for the SGRA problem is given in Fig. 4.

D. Parallelization with Hybrid OpenMP/MPI Programming

Hybrid applications use both the OpenMP and the MPI models together for parallelism. A hybrid model requires a more sophisticated programming paradigm than either type of the constituents so as to manage shared and distributed memory allocations with multiple cores. We aim to use a higher number of cores in the HPC system and reduce the computation time. It is worthwhile to mention again that we are not dealing with the technical challenges of using these parallelization methods. In this paper, we are simply aiming to provide information on basic programming models and

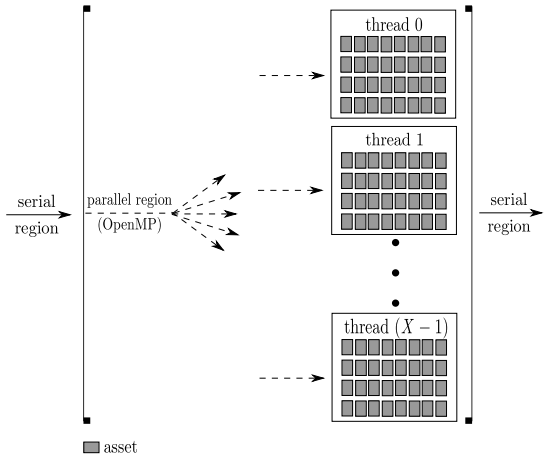

Fig. 2. OpenMP implementation of SGRA (X cores).

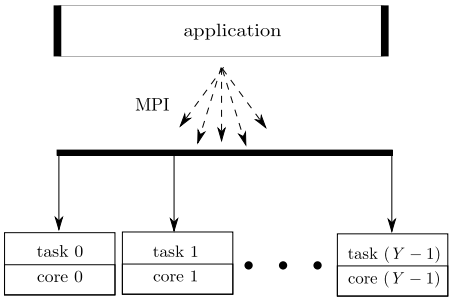

Fig. 3. MPI application utilization model.

compare their performances in terms of computation time. In Fig. 5, the hybrid parallelization model for the SGRA study is presented.

With the hybrid model, Y task is created for separating days and X threads are used for each task to distribute the smart appliances (assets) among threads to determine the aggregator profit for each day. In this way, $X \times Y$ cores are used with hybrid parallelization. For example, consider that five MPI tasks are defined and each task is deployed on three OpenMP threads to solve an optimization problem, thus totally 15 cores are utilized at the same time.

IV. Simulation Results

A. HPC Platform

For the SGRA application, we used the Summit Colorado State University and the University of Colorado Boulder HPC system [18], [19]. Summit is a heterogeneous supercomputing cluster with 380 Haswell CPU nodes with 9,120 cores, ten GPU nodes, five hi-mem nodes, two storage gateway nodes, two Omnipath Architecture (OPA) interconnect fabric management nodes, with 100 GB/sec OmniPath interconnect, one Petabyte DDN SFA14K scratch storage, and a 40Gb uplink to the Science Ethernet Network. The Summit HPC uses a batch queuing system for execution. Finally, the SGRA problem was programmed in C++ and compiled with the gnu-c++-compiler on Summit.

B. System Setup

The above-mentioned parallel processing techniques for the SGRA problem are performed on a test system with 5,555 customers, 56,605 controllable assets, and 151,773 base loads [12]. The daily electric energy consumption of controllable appliances, i.e., assets, is obtained from the probabilistic model given in [12] and base loads consumption is determined from the data in [20]. To obtain a fair comparison of the performance of the parallel processing techniques, we used the same set of electricity profiles for seven days (the length of multi-days SGRA) in each parallel processing technique. The simulations are carried out for seven days at a resolution (simulation time step) of 15 minutes. For the spot market and utility electricity prices, real data corresponding to July 1-7, 2011 are used from PJM and ComEd, respectively [13], [14]. The price of electricity for the desired number of days are automatically retrieved from the PJM and ComEd websites using a Python script. The hourly price data are available on the respective websites in HTML or Zip file format. The Python script is executed to fetch the price data for each day and convert them to CSV format to match the simulation time step (resolution). For example, the hourly price data is reconstituted to a 15-minute time resolution for the simulation process by holding the hourly price value as constant. Thus, a database comprised of the predetermined energy consumption profile of the appliances and the electricity price information is created for the entire simulation run time duration corresponding to multiple days. Finally, the SGRA control algorithm utilizes this database as input for executing each of the three parallel processing methods.

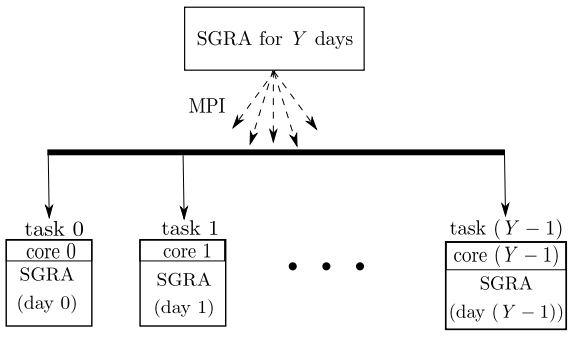

Fig. 4. MPI implementation of SGRA (Y cores).

To solve the SGRA problem, which is heuristic in nature, Genitor—a modular genetic algorithm (GA) package [21]—is used as an optimization solver. Genitor creates two children from the initially generated population (100 members) at each iteration by applying crossover and mutation operations. We assume that the optimization is completed when the result of the best fitness function does not change for 10,000 consecutive iterations or if the total number of iterations reaches 500,000.

C. Performance Evaluation

In this section, the performances of the above-mentioned three parallelization methods are compared for different numbers of OpenMP threads and MPI tasks. The results are given in term of the total computation time (in hours) and per-iteration-time of GA (in seconds). Because the GA is a heuristic optimization method, it generates different near-optimal results for each run due to generating new individuals probabilistically. Therefore, for establishing a fair comparison of the performance of the parallelization techniques, the computation time between two successive iterations of the GA is chosen.

In Fig. 6, the total and per-GA-iteration computation times are given for a number of OpenMP threads. The computation time is recorded over the range of (6, 15) hours; where the maximum time occurs when one thread (i.e., the base case with one core) is used and the minimum time of 6.5 hours occurs when seven threads are used; thus, a 60.24% reduction in computation time is achieved. Note that there is

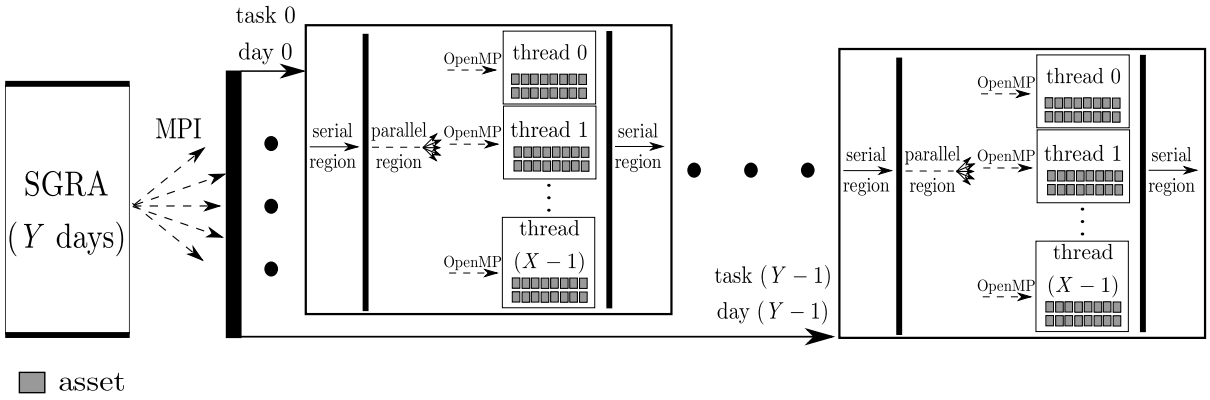

Fig. 5. OpenMP plus MPI implementation of SGRA ($X \times Y$ cores).

an exponential relation between the computation time and the number of OpenMP threads. Therefore, the biggest reduction in computation time between two OpenMP cases–43.12%–is achieved among thread numbers one and two. Beyond that, when the thread number is increased from two to seven, only 17.12% additional reduction in computation is achieved as compared to the base case.

However, as mentioned before, the reduction in computation time can be misleading for accurate comparison because of the difference in total number of iterations. When the computation times of per-GA-iteration are compared, a 57.74% reduction compared to one thread in computation time is achieved with seven threads. It should be noted that more than seven threads—up to a theoretical maximum number equaling the assets—can be defined for the studied case. For fair comparison with the MPI case, we limited the number of threads for this study to seven.

In Fig. 7, the results of the MPI model for a varying number of tasks are given. The minimum simulation time recorded was 197 minutes when seven MPI tasks are utilized. Here, we see a reduction of 79.57% in simulation time and 77.56% in computation time of per-GA-iteration achieved. This indicates a better performance when compared to the previous case of OpenMP with seven threads. In other words, when the same number of cores are considered, MPI outperforms the OpenMP in executing the SGRA problem due to the absence of any serial parts. Note that we used a maximum of seven MPI tasks to match the number of days in the multi-day example that defined in our test case.

In Fig. 8, the results of the hybrid OpenMP/MPI model are presented and compared for a varying number of OpenMP threads and MPI tasks combinations. When combining the two methods in a hybrid mode, more cores can be used (such as seven threads $\times$ seven tasks) with greater efficiency based on the problem formulation. In our case, the core numbers can be increased when using the OpenMP only, which is less efficient than MPI. On the other hand, the number of cores used by the MPI to solve the SGRA problem is defined by the number of days in the multi-days SGRA. For instance, even though there are 9,120 general purpose Haswell CPU cores available in the Summit HPC, the multi-day SGRA problem (for seven days) uses a smaller subset of those cores (i.e., a maximum of 49 cores). In Fig. 8, the same SGRA problem from the above examples is solved in 100 minutes—corresponding to a 89.46% reduction using 49 cores. By using 42 extra cores, approximately 10% additional simulation time reduction is achieved as compared to the seven MPI tasks-one OpenMP thread case (see Fig. 7(a)).

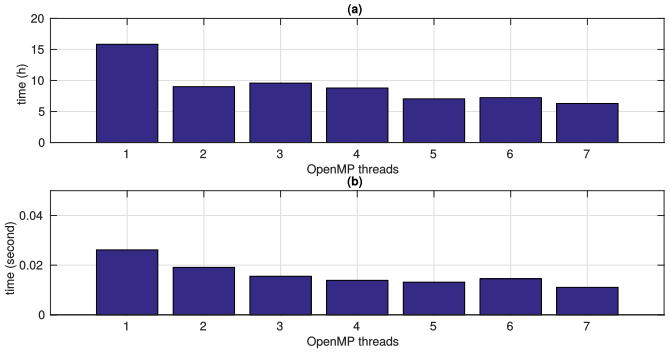

Fig. 6. Number of OpenMP threads versus (a) total simulation time and (b) computation time for per-GA-iteration.

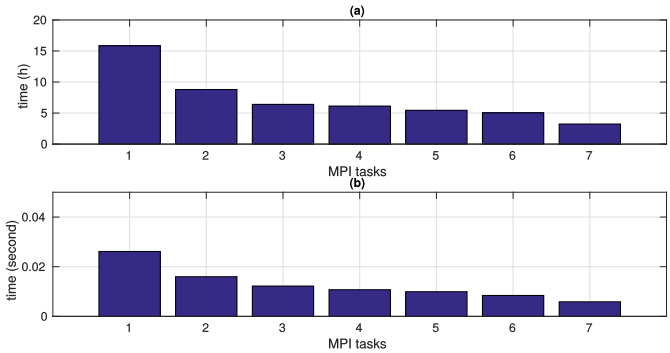

Fig. 7. Number of MPI tasks versus (a) total simulation time and (b) computation time of per-GA-iteration.

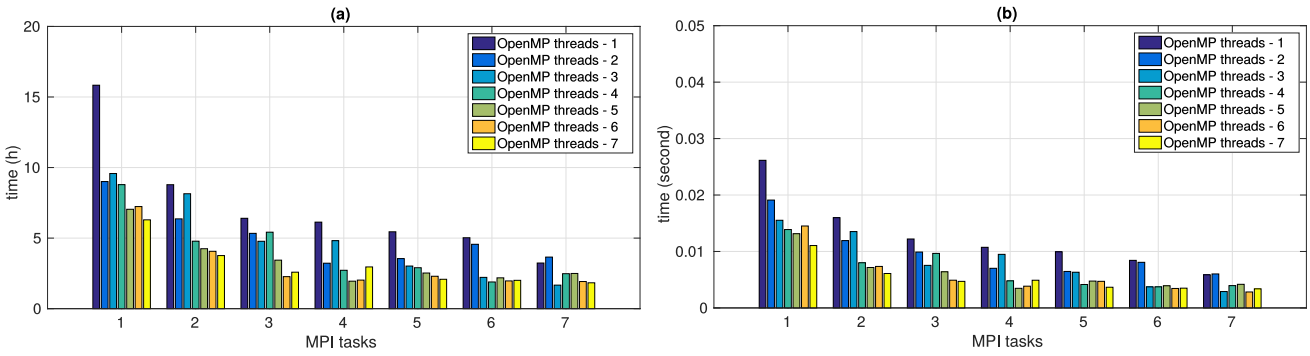

Fig. 8. Number of hybrid threads and tasks versus (a) total simulation time and (b) computation time of per-GA-iteration.

The results presented cannot be generalized; however, they provide an insight into the use of the three parallelization methods for the SGRA problem. As we embark on the task of parallelizing the SGRA problem for larger test systems for longer time horizons, this effort should serve as an early indicator for the choice of the technique.

V. Conclusion

In this paper, three parallel processing methods—OpenMP, MPI, and hybrid OpenMP/MPI—are presented to reduce the simulation time of an optimization problem in the Smart Grid realm. Parallelizations are implemented on the SGRA test case for multiple days on the Summit HPC system. The performance of the methods are quantified in terms of reductions in the simulation time and computation time of per-GA-iteration, and the results are compared against the number of cores utilized. We conclude from the results that all parallelization methods can significantly affect the computation time for solving the SGRA problem. We also note that the computation time appears to fall exponentially with the number of cores utilized. Among the parallelization methods, the hybrid OpenMP/MPI model showed the best performance when a higher number of cores are available and the MPI model was the second best. However, it should be noted that performance of the methods are highly dependent on the programming of the optimization problem.

References

[1] Federal Energy Regulatory Commission. FERC Staff Issues Assessment of Demand Response and Advanced Metering. [Online]. Available: https://www.ferc.gov/industries/electric/indus-act/demand-response/dem-res-adv-metering.asp. [Accessed Apr. 2017]

[2] A. Zipperer, P. A. Aloise-Young, S. Suryanarayanan, R. Roche, L. Earle, D. Christensen, P. Bauleo, and D. Zimmerle, "Electric Energy Management in the Smart Home: Perspectives on Enabling Technologies and Consumer Behavior," in *Proceedings of the IEEE*, vol. 101, no. 11, pp. 2397–2408, 2013.

[3] B. Celik, R. Roche, D. Bouquain, and A. Miraoui, "Coordinated Energy Management Using Agents in Neighborhood Areas with RES and Storage," in *2016 IEEE International Energy Conference (ENERGYCON)*, 2016, pp. 1-6.

[4] L. Gkatzikis, I. Koutsopoulos, and T. Salonidis, "The Role of Aggregators in Smart Grid Demand Response Markets," *IEEE Journal on Selected Areas in Communications,* vol. 31, no. 7, pp. 1247-1257, 2013.

[5] A. J. Conejo, J. M. Morales, and L. Baringo, "Real-time Demand Response Model," *IEEE Transactions on Smart Grid,* vol. 1, no. 3, pp. 236-242, 2010.

[6] D. Setlhaolo, X. Xia, and J. Zhang, "Optimal Scheduling of Household Appliances for Demand Response," *Electric Power Systems Research*, vol. 116, pp. 24-28, 2014.

[7] P. Faria, Z. Vale, J. Soares, and J. Ferreira, "Demand Response Management in Power Systems Using Particle Swarm Optimization," *IEEE Intelligent Systems,* vol. 28, no. 4, pp 43-51, 2013.

[8] R. Roche, S. Suryanarayanan, T. M. Hansen, S. Kiliccote and A. Miraoui, "A Multi-Agent Model and Strategy for Residential Demand Response Coordination," in *2015 IEEE Eindhoven PowerTech,* 2015, pp. 1-6.

[9] J. S. Vardakas, N. Zorba, and C. V. Verikoukis, "A Survey on Demand Response Programs in Smart Grids: Pricing Methods and Optimization Algorithms," *IEEE Communications Surveys & Tutorials,* vol. 17, no. 1, pp. 152-178, 2015.

[10] A. Stamatakis, and M. Ott, "Exploiting Fine-Grained Parallelism in the Phylogenetic Likelihood Function with MPI, Pthreads, and OpenMP: A Performance Study," in M. Chetty, A. Ngom, S. Ahmad (eds), Pattern Recognition in Bioinformatics. PRIB 2008. Lecture Notes in Computer Science, vol 5265. Springer, Berlin, Heidelberg.

[11] H. Brunst, and B Mohr, "Performance Analysis of Large-Scale OpenMP and Hybrid MPI/OpenMP Applications with Vampir NG," in M.S. Mueller, B.M. Chapman, B.R. de Supinski, A.D. Malony, M. Voss(eds) OpenMP Shared Memory Parallel Programming. Lecture Notes in Computer Science, vol 4315. Springer, Berlin, Heidelberg.

[12] T. M. Hansen, R. Roche, S. Suryanarayanan, A. A. Maciejewski, and H. J. Siegel, "Heuristic Optimization for an Aggregator-Based Resource Allocation in the Smart Grid," *IEEE Transactions on Smart Grid,* vol. 6, no. 4, pp. 1785-1794, 2015.

[13] *PJM Daily Real-Time Locational Marginal Pricing (LMP).* [Online]. Available: https://goo.gl/r5ppJk. [Accessed Apr. 2017]

[14] *ComEd Real-Time and Day-Ahead Hourly Prices.* [Online]. Available: https://hourlypricing.comed.com/live-prices/. [Accessed Apr. 2017]

[15] OpenMP Application Program Interface, *OpenMP Architecture Review Board.* [Online]. Available: http://www.openmp.org/specifications/. [Accessed Apr. 2017].

[16] C. T. Yang, C. L. Huang, and C.F. Lin, "Hybrid CUDA, OpenMP and MPI Parallel Programming on Multicore GPU Clusters," *Computer Physics Communications,* vol. 182, pp. 266-269, 2011.

[17] MPI: A Message-Passing Interface Standard, *Message Passing Interface Forum.* [Online]. Available: http://mpi-forum.org/docs/. [Accessed Apr. 2017].

[18] Summit - High Performance Computing System: User's Guide, *The Information Science & Technology Center - Colorado State University.* [Online]. Available: https://www.acns.colostate.edu/hpc/. [Accessed Apr. 2017].

[19] *Summit: Research Computing - University of Colorado Boulder.* [Online]. Available: https://www.rc.colorado.edu/resources/compute/summit. [Accessed Apr. 2017]

[20] R. Roche, "Agent-Based Architectures and Algorithms for Energy Management in Smart Grids," *Ph.D. dissertation, Sciences Pour l'Ingénieur et Microtechniques, Institut de Rechecrche sur les Transports, l'Energie et la Société – Laboratoire Systémes et Transport, Univ. Technol. Belfort-Montbéliard,* Belfort, France, Dec. 2012.

[21] D. L. Whitley. *Genitor: Modular GA Package with Floating-Point Support.* [Online]. Available: https://goo.gl/i8kPCg. [Accessed Apr. 2017]