Received 6 July 2023, accepted 31 July 2023, date of publication 7 August 2023, date of current version 14 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3303282

RESEARCH ARTICLE

Parallel Implementation of a Sailing Assistance Application in a Cloud Environment

MARCIN ZYCZKOWSKI[1], RAFAL SZLAPCZYNSKI[1], PIOTR ORZECHOWSKI[2], AND HENRYK KRAWCZYK[2,3]
[1]Faculty of Mechanical Engineering and Ship Technology, Gdańsk University of Technology, 80-233 Gdańsk, Poland
[2]TASK Informatics Center, Gdańsk University of Technology, 80-233 Gdańsk, Poland
[3]Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, 80-233 Gdańsk, Poland

Corresponding author: Marcin Zyczkowski (marcin.zyczkowski@pg.edu.pl)

This work was supported in part by the Pomorskie Voivodeship Regional Operational Program, from 2014 to 2020, under Grant RPPM.01.02.0022-0001/17 and Grant CK STOS.

ABSTRACT Sailboat weather routing is a highly complex problem in terms of both the computational time and memory. The reason for this is a large search resulting in a multitude of possible routes and a variety of user preferences. Analysing all possible routes is only feasible for small sailing regions, low-resolution maps, or sailboat movements on a grid. Therefore, various heuristic approaches are often applied, which can find solutions within an acceptable time, sacrificing their optimality and accuracy. In this study, we propose a different approach based on the parallel implementation of an exact algorithm. Specifically, we present a Sailing Assistance Application (SAA) utilizing a deterministic approach and show how it can be parallelized in a cloud environment to reduce its execution time. The potential of the proposed parallelization method goes beyond the particular presented solution; it can be used to improve the performance of other weather routing tools such as collision avoidance and related applications.

INDEX TERMS Big data, cloud computing, constraint optimization, decision support systems, graph theory, marine navigation, parallel architecture, path planning.

I. INTRODUCTION

Route planning in maritime transport has inspired researchers in the era when sailing ships were the only means of transport that enabled transoceanic travel. The first recommended a route for ocean shipping was made by Maury [1] in the 19th century. Currently, in sea shipping, a voyage plan is mandatory in accordance with the International Convention for the Safety of Life at Sea (SOLAS), which imposes the requirement of planning the ship's route before starting the voyage [2]. The voyage-planning obligation applies to all types of vessels, including sailing vessels. When planning a route for sailing vessels, special attention is paid to meteorological conditions, particularly wind conditions, which are the main determinants of the navigation possibilities. Moving directly against the wind is unacceptable and even forbidden because of the possibility of damage to sails.

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Marozzo.

Currently, data on meteorological conditions are disseminated in a publicly available, cyclical manner, with increasing frequency, longer forecasts, and greater resolution [3]. This is extremely useful for a safe navigation. An increased frequency of dissemination of meteorological data increases the reliability of meteorological forecasts. Moreover, a greater resolution of meteorological data increases the accuracy of marine-weather forecasts. This new approach, which disseminates meteorological data through weather services, provides new opportunities for decision support systems in navigation. It must be emphasized here that the digital description of the sea area is as important as weather forecasts. The larger the set of points describing the sea area, the greater the accuracy of the description of this area. The increased volume of data describing hydro-meteorological conditions and sea area increases the possibility of ensuring safety by systems supporting maritime navigation. However, this means an increased amount of data is processed by the computer, which in turn increases the computational time.

Therefore, this paper presents a method for finding the optimal route for a sailing vessel that has been implemented using parallel programming on many cores simultaneously. The current sequential version of the Sailing Assistance Application (SAA) is available on a typical server [4]. The proposed modification is oriented towards parallel programming and cloud computing to reduce the waiting time significantly for the result of SAA. For this reason, the application can be used not only for route planning, but also for sailing along the planned route. Moreover, the new sailing data update is significantly simplified through the use of specialized cloud services. The proposed approach uses a multithread computer architecture available in the TASK (Tricity Academic Supercomputer and networK) Informatics Center, and TASKcloud implemented in that Center [5].

The original SAA application is a sequential one, where the algorithm is a sequence of steps. The algorithm takes inputs from the user and, after some computations, produces an output. We aim to transform sequential SAA components into parallel ones, where data parallelism is implemented. Data parallelism involves running the same task on different components of data. In other words: the same computation is done for every thread, but we feed those threads with different parts of the data. and next the model is split among threads. In consequence, our main task is to transform sequential SAA code into a parallel one using an adequate parallel language and platform. Moreover, to test to obtain solution, a cloud environment is prepared. Our solution is based on open-source platforms, such as OpenMP and OpenStack.

Related studies are presented in the next section. Subsequently, the components of the SAA are described, and a general optimization problem is defined. Parallelization methods are then discussed, and the cloud testing environment is characterized. Finally, the simulation results are provided, and the conclusions are formulated.

II. RELATED WORKS

The presented research problem concerns computer support for voyage planning of sea-going ships. There are several categories of solutions in the literature on the subject: Autonomous Surface Vehicles (ASV), Unmanned Surface Vehicles (USV) [6], [7], [8], [9], conventional power-driven vessels [10], [11], [12] and sailing vessels [13], [14], [15]. They differ not only in the characteristics of the objects used, but also in the strategy of proceeding with the implementation of the adopted tasks. Nevertheless, two main approaches to the problem of route planning can be distinguished: deterministic and nondeterministic. Deterministic methods are more time-consuming, but they are also characterized by high repeatability of the results. The solutions proposed in [13], [16], and [17] correspond to such an approach. In general, they are based on path analysis in a graph representing the sailing area. Non-deterministic methods include solutions using various types of swarm or evolutionary algorithms [18], [19]. Indeterministic methods accentuate efficiency; however, they do so at the expense of optimality and repeatability. Therefore, deterministic methods are still superior to the latter in terms of quality of results. Considering the above, it is both desirable and challenging to reduce their high computational time and computational memory.

In the case of SAA, we were interested in adopting a weather routing algorithm for parallel processing requirements. Similar objectives have been pursued by other researchers in the field, who have also approached the problem of parallelizing complex optimization algorithms. These studies include deterministic and swarm algorithms for path planning [20], [21], [22], [23], graph distance computations [24] and optimization of robot motion [25], [26]. However, for weather routing of ships, sequential solutions still dominate. In particular, no parallel approach to weather routing of sailboats has yet been proposed. This study aimed to fill this gap.

To achieve successful parallelization, we must use appropriate parallel mechanisms and platforms. Representative examples of such solutions are Message Passing Interface (MPI) [27] primarily oriented towards distributed processing between the nodes of a computer system, or Open Multi-Processing (OpenMP) [28] designed for multithreaded processing within a single node or the OpenCL standard, which offers a common API for program execution on systems composed of different types of computational devices such as multicore CPUs, GPUs, or other accelerators [29]. The above solutions offer mechanisms that can be used to either create parallel programs or transform sequential programs into parallel programs.

Nowadays, the multi-core processors, based on including more processing cores and a shared cache memory in a single microprocessor, are gradually becoming more prominent in the commercial market. Therefore, in this study, we focused on multithread processing. In particular, OpenMP provides a fork-and-join execution model, in which a program begins execution as a main thread. This thread is sequentially executed until a parallelization directive for a parallel region (e.g., for a loop) is found. The main thread creates then a team of threads and controls their operations. All the threads execute their statements and at the end of the parallel constructs, all of them are synchronized. The essential advantage of OpenMP is that an existing code can be easily parallelized by placing OpenMP directives around time-consuming loops which do not contain data dependencies. However, optimizing work-flow and memory access is not a trivial issue, when using OpenMP, and such aspects as data partitioning, functional dependencies, size of the parallel loop, etc. must be controlled by the programmer.

Because both the application components and the used algorithms were important, a real multithread environment was required to test them. TASKcloud [5], [30] was chosen for this study because it satisfies the general requirements of cloud computing [31] and allows the use of service software engineering paradigmatic [32]. It is built from open-source components, such as OpenStack [33] and Docker [34] offering flexible solutions owing to the container mechanisms. In this environment, SAA was investigated to determine the possibility of parallelization of some of its main elements. We concentrated on all the components of the SAA and attempted to find the solution leading to a reduction in the required processing time. The simulations showed that the proposed parallelization technique resulted in a nearly 80% reduction in computational time.

Until now, there were no work on parallelization of weather routing for sailboats. In particular, these authors' earlier ship weather routing-related research was not based on parallel programming. As has been argued above, the proposed approach to the problem makes it possible to reduce the working time of the algorithms significantly, without sacrificing the method's repeatability and reliability in terms of finding optimal solutions for the considered criteria.

To find optimal route, some parallel Dijkstra algorithms have been proposed in the literature, but the majority of them consider input data set described by matrix and are tested rather for small data size (hundreds of elements instead of a million) [27]. Moreover, in our case, more complicated optimization criteria (e.g., manoeuvres penalty) and some dynamic parameters (e.g., weather forecasting) are considered. Additional motivation was to create a solution useful as a real-world application.

III. THE IDEA BEHIND THE SAA

We can distinguish three components of SAA, which perform the following functions:

1) Creating a navigation area – the point of a geographical area (bit map)–is analysed and modelled by the points creating the area grid with the required resolution determined with a single rectangle representing one step of navigation. The entire matrix representing the grid area was generated, and its navigable and non-navigable points were identified.
2) Establishing sailing conditions – a graph model of the grid area is created, and a set of data representing current sailing conditions is assigned to each vertex of the graph. It is created utilizing available ship characteristics and weather forecast data in the GRIB2 file format. GRIB2 is a file format for the storage and transfer of gridded meteorological data, such as Numerical Weather Prediction model output [3]. Based on the above, the time required to sail between two neighbouring points was estimated. Other characteristics of the sailing route were also determined.
3) Finding route – using well-known pathfinding algorithms and considering the user requirements (e.g., the shortest sailing time, highest comfort of travel, and acceptable level of safety), the best acceptable route from start point S to finish point F for the considered graph is determined. First, SAA uses a generalized Dijkstra's algorithm, which handles graphs of dynamically changing edge weights.

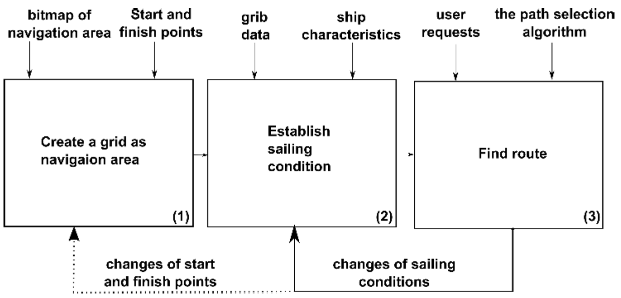

bitmap of navigation area
Start and finish points
grib data
ship characteristics
user requests
the path selection algorithm
Create a grid as navigaion area
(1)
Establish sailing condition
(2)
Find route
(3)
changes of start and finish points
changes of sailing conditions

FIGURE 1. Main diagram of SAA architecture.

In Fig. 1, the structure of the SAA is given, where the above components are presented, and the required main input data are listed.

In practice, component 1 is the appointed navigation region limited by lines of min and max latitudes and min and max longitudes, denoted by φ_1, φ_m, λ_1, and λ_n, respectively. The total number of area points, denoted by, p_{total} is as follows:

$$p_{total} = \frac{(\varphi_m - \varphi_1)\left(\lambda_n - \lambda_1\right)}{d_{lon}} = mn \tag{1}$$

where: m, n – numbers of horizontal and vertical curves, respectively, φ_1, φ_m, λ_1, and λ_n are the minimum and maximum values of the geographical coordinates of the area, d_{lat}, d_{lon} – vertical and horizontal distances between neighbouring points, determining the degree of area resolution, that is, $g = d_{lat} \times d_{lon}$

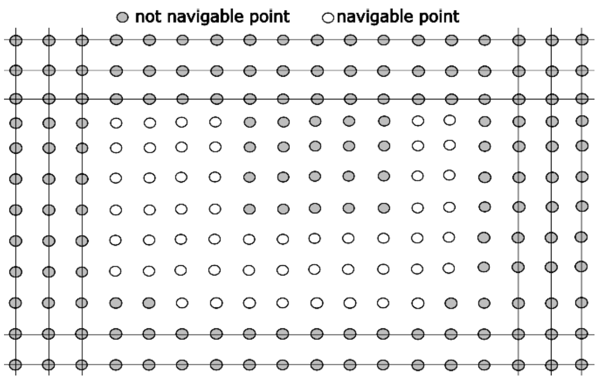


FIGURE 2. The appointed navigation region with navigable and not navigable points.

Taking into account the bathymetry of the area, ship submersion, and headroom under the keel, the shape of the navigation area can be clearly specified by the navigation points, and can be different from regular shapes, such as rectangles or squares. A bitmap of such a region can be generated automatically, and the points of the grid are classified (with labels of 0 or 1) as either acceptable or not acceptable for sailing. Each route consisted only of acceptable sailing points. To reduce the size of the grid, all rows, and columns containing only not acceptable points can be cancelled. Then, the real number of points in the grid that need to be taken into consideration is less than that estimates above (see Fig. 2).

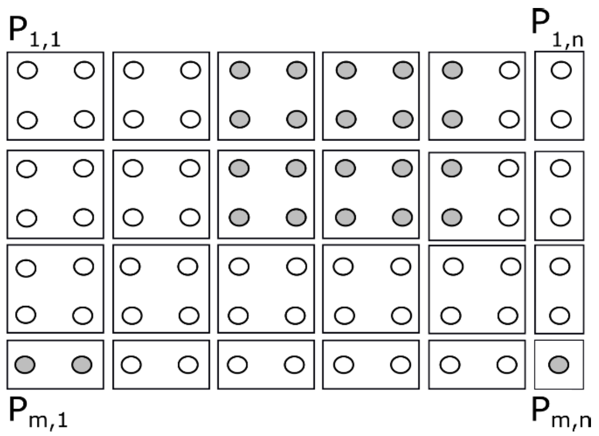


FIGURE 3. The weather forecasting data (GRIB) for the reduced navigation region. Mostly, on this figure, one GRIB consists of four points of grid.

Consequently, the navigation area is determined by the set of points (see Fig. 3) representing the reduced grid:

$$P_{ij} = P\left(\varphi_i, \lambda_j\right) \tag{2}$$

where:

φ_i– latitude value, λ_j – longitude value

$i = 1, 2, \ldots, m; j = 1, 2, \ldots, n.$

The up-left corner of this area corresponds to point. P_{11}

Let us consider such a sailing area. Component 2 of SAA enriches it with some specific data, where current sailing conditions were assigned to each point. P_{ij} First, the weather conditions (temperature, wind direction, and waves' height) were registered and should be stored for each point based on the GRIB file. Moreover, weather services update information periodically, e.g., every six hours, then SAA traces it and changes all information assigned to the grid. This implies that the weather updates can be included several times during a sea voyage.

Another condition under consideration is the vessel characteristics essential for ship manoeuvring (available from the VPP sailing vessel). These can also be merged with the general sailing conditions stored in the considered grid points.

Based on the current information regarding all conditions discussed above, the time of sailing between neighbouring points is re-estimated. Moreover, the weather conditions can be registered with higher area resolution than the navigation area is done. Then for subsets of the grid points extra data interpolation must be performed to describe the sailing conditions, what is shown in Fig 3. We can see that the same weather conditions are assigned to at most four points of the navigation area.

To consider the possible vessel routes, the limited number of navigation directions should be determined by the size of the considered rectangle, where the considered point is its center (see Fig. 4). In general, if all movement directions are considered, we require a rectangle with nine points to show eight direction possibilities. For 16 possibilities, we should consider rectangles consisting of 25 points,

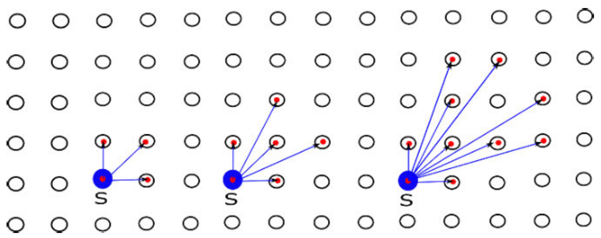


FIGURE 4. The example: a sailing area with three different starting points: P2 3, P2 6, P2 10, and three possibilities of some different movement directions.

and for 32 possibilities, we should consider rectangles of 81 points. The number of edges in the grid depends on the assumed number of movement directions for each grid point. Let d denote the vertex degree, that is, the number of edges emerging from each vertex of a grid (8, 16, 32, etc.). It also denotes the assumed number of navigational directions. In general, d *and* g were constant throughout the optimization process. However, we can imagine a strategy in which d *and* g can be changed. For example, sailing conditions are represented more precisely for some areas of a grid (in direct proximity to land or other constraints).

The maximal number of edges in the grid is equal to:

$$e_{max} \leq p_{\text{total}} \cdot v \cdot d = m \cdot n \cdot d. \tag{3}$$

$V\left(P_{ij}\right)$ is the set of neighbouring points P_{kl} corresponding to the assumed directions. The vertices with red dots in Fig. 3 show only such directions.

Based on the registered information for each neighbour points, we can estimate independently corresponding vessel movement time between this pair of points. It depends on such parameters as: vessel movement direction, vessel's speed, wind direction, waves' height [35]. Besides, we must consider vessel manoeuvres described by turning angle because they also affect total sailing time. In consequence, we must take into account three positions of the vessel: previous (P_{k-1}), current (P_k) and next one. (P_{k+1}) This allows us to calculate the turning angle and to estimate the time of vessel movement τ according to formula (4).

$$\begin{aligned} \tau\left(P_{k-1}, P_k, P_{k+1}\right) &= \mid \propto_{k-1,k} - \propto_{k,k+1} \mid \cdot t_{\Delta\alpha} \\ \tau_k &= \mid \propto_{k-1,k} - \propto_{k,k+1} \mid \cdot t_{\Delta\alpha}, \end{aligned} \tag{4}$$

where:

$\propto_{k-1,k}$ is course over ground from, P_{k-1} *to* P_k

$\propto_{k,k+1}$ is course over ground from, $P_k to P_{k+1}$

$t_{\Delta\alpha}$ is a time-equivalent penalty delay for a given course change, expressed in seconds per degree. It directly follows from characteristics of the vessels (for example SV Conrad [16]) and it is calculated when manoeuvre of the vessel is desirable. In consequence, it increases the vessel movement time between the neighbour points: P_k and P_{k+1}. Let express briefly: $\tau(P_k, P_{k+1}) = \tau\left(P_{k-1}, P_k, P_{k+1}\right), \tau\left(S, P_{k+1}\right) = 0$, $\tau(P_k, F) = 0$

The grid, whose points contain relevant information, is used for determining the most acceptable sailing route between the given S and F points. The process is as follows. A user of the SAA chooses the sailing region and specifies the S and F points, as well as the starting date and time.

To calculate the optimal route (component 3), a grid model was transformed into a graph model. We consider a weighted graph, $G = (V, E, \omega(e, \tau(e)))$ where V is the set of vertices, E is the set of edges, $e_{ij} = (v_i, v_j)$ means that it is possible for a ship to move from v_i to, v_j, $e_{ij} \in E$. Then ω is a function that assigns to edges all weights given in advance and calculated. Edge weights may be determined based on the sailing time (having regards to manoeuvres – formula (4)), sailing cost, or any other quantity that accumulates additively along a path.

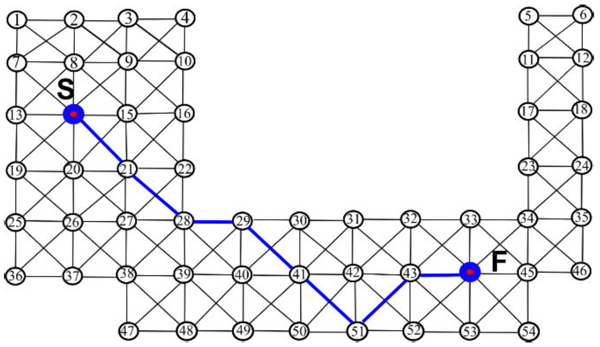


FIGURE 5. Example of graph G(V,E) and a final route from point S to point F as result of component 3 run.

In our case, V represents all navigable points of the grid, and E contains arcs representing all the admissible movement directions for each of the navigable points. Fig. 5 shows a graph corresponding to the grid represented in Fig. 3. It is worth emphasizing that the graph does not include external non-navigable points. Each edge represents movement into directions.

Once some weights are assigned to the graph edges, the task of component 3 can be started. For the clarity of the description, it is assumed that the vertices in the graph are numbered lexicographically from the up-left corner to the bottom right.

The optimization problem can be presented as finding a sequence of graph edges between route endpoints, to minimize a given goal function. Let a route $R = (e_1, e_2, \ldots e_i, e_{i+1}, \ldots e_r)$ be the path (sailing route) from S to F, where for $e_i = (v_x, v_y) \epsilon E, e_{i+1} = (v_y, v_z) \epsilon E, v_y, v_z \epsilon V$. Fig. 5 presents the following route: ((14, 21), (21,28), (28,29), (29,41), (41,51), (51,43), (43,44)); r = 7. In general, the number of vertices in route R is equal to r+1, the number of edges between them equals r.

According to the proposed model, we try to find the route R satisfying the following criteria:

$$\sum_{e \epsilon R} \omega(e, \tau(e)) = \sum_{i=1}^{r} \omega(e_i, \tau(e_i)) | e_i \in R = min \quad (5)$$

for all possible routes from S to F in the graph $G = (V, E, \omega(e, \tau(e))$. As can be seen, the problem is described by a changeable number of decision variables – edges within a route. Depending on the sailing conditions and ship characteristics, the values of weights $\omega(e, \tau)$ assigned to all edges can be determined for all $e \in E$, as shown in [36]. Following this, Dijkstra's algorithm and other methods can be used to determine the optimal route. Assuming that the complexity of a single read/write operation, single label assignment operation and the single operation related to graph weight calculations are all equal to $O(1)$, the sequential time complexities of the algorithms of the presented components are as follows [37], [38]:

Component 1: $O(m \cdot n)$

Component 2: $O(m \cdot n \cdot d)$

Component 3: $O[(m \cdot n \cdot d + m \cdot n \cdot d \cdot log(m \cdot n)] = O[m \cdot n \cdot (1+d\, log(m \cdot n)] = O[m \cdot n \cdot d \cdot log(m \cdot n)]$.

IV. PARALLELIZATION METHODS AND CLOUD TESTING ENVIRONMENT

The objective of parallelization is to minimize the time of SAA execution, particularly if either a sailing region is too large or a high grid resolution is applied. One possible the approach is to replace exact algorithms with heuristic algorithms. However, this may lower the quality of the obtained results, which is not acceptable, even if such algorithms are easier to parallelize [39]. Despite this, in this study, we focused on the data parallelization of each component of the SAA. To achieve this goal, we divide the set of data into several parts and calculate each part independently. Additionally, we turn the sequential version of the SAA into a parallel version using OpenMP programming platform. The functional parallelization approach used here relies on splitting the program into subprograms (components responsible for various tasks), which can be executed concurrently. Because the computing node consists of many processing units (cores), subprograms can be assigned to the cores and executed in parallel. Determining the efficiency and thread mapping depends on the type of architecture (shared memory multicore, memory hierarchy, and operating system) and the type of application to be run (types of components, required data, and various time dependencies). Unfortunately, there is no single-thread mapping strategy that suits all the possible applications. Any OpenMP program begins as a single process called the master thread. When the master thread reaches the parallel region, multiple threads are created to execute the parallel codes enclosed in the parallel region. When the threads complete the parallel region, they synchronize and terminate, leaving only the master thread. We initiated the OpenMP programming model with a simple program including the directive *#pragmaomp parallel* [clause list]. First, it declares the variables in the list to be shared among all threads in a team, and variables in the list to be private to each thread in a team. Second, it uses loops, where each iteration of the loop is assigned a different thread, and sets the dynamic schedule for computational resource threads. In other words, we use such mechanisms to announce to the compiler that the next code block should be executed as a new task. The execution of the tasks can be instant or delayed according to the task schedule and availability of the threads. This means that there is no known dependency a priori between the task execution time and the number of engaged threads. This can only be determined empirically.

The above approach was used separately for each component of the SAA. To *Creating a navigation area,* we can analyse some parts of the bitmap in parallel to create a grid of sailing areas. Similarly, we can create a digraph G by *Creating a navigation area* and estimating its weight values. All calculations for the given vertices were divided between threads.

Consider the graph in Fig. 5. For each edge of the graph, we read the four parameters as follows:

x_0– vessel movement direction

x_1– vessel's speed,

x_2 – wind direction,

x_3 – waves' height.

```
i)
for $\forall v \in V$ {
  for $\forall e \in E^*(v)$ {
 /$E^*(v)$  represents all edges outcoming from v not having calculated weights/

    for (r = 0; r <3; ++r) {
      read ($x_r$)
    }
   calculate $w_e$
   assign $w_e$ to $e \in E^*(v)$
  }
}

ii)
# pragma omp parallel for
shared ($x_i$, $w_e$) private (s)
for (int s = 0;   s < | V |; ++s) {
  collect data, calculate and assign $w_e$ to e
}
```

FIGURE 6. Example of sequential (i) and parallel (ii) the considered pseudocode.

Based on the above values, we can calculate $\omega(e, \tau) = F(x_0, x_1, x_2, x_3)$. In practice, is a complex procedure that has been analysed in many scientific papers [4], [16], [17]. In Fig. 6, we show the concept of parallelization for such hypothetical calculations. To simplify the pseudocode, we assumed that the internal loop in the sequential code is presented as a general procedure in parallel.

The parallel pseudocode contains a special directive that instructs the compiler to generate a code that splits the iterations of the external loop among multiple threads. If the number of threads is three, splitting can be performed as follows:

Thread 1: the number of chosen vertices; 1; 4; 7; 10; 13; 16; 19; 22; 25; 28; 31; 34; 37; 40; 43; 46; 49; 52;

Thread 2: the number of chosen vertices: 2; 5; 8; 11; 14; 17; 20; 23; 26; 29; 32; 35; 38; 41; 44; 47; 50; 53;

Thread 3: the number of chosen vertices; 3; 6; 9; 12; 15; 18; 21; 24; 27; 30; 33; 36; 39; 42; 45; 48; 51; 54;

where the integers represent the number of vertices in a graph G given in lexical order (see Fig 5). Below is a diagram of the division of tasks into individual threads from T1 to T3 in the graph, where the number of all vertices is 54.

Note that in the case of data parallelism, when we consider h disjoint data blocks, the above formula existing in the brackets must be divided by h. In general, $h \ll m{\cdot}n$, which means that parallel computational complexity is on the same level.

In the case of Component 3, the input data is representing by weight *graph* $G = (V, E, \omega)$. We can consider separate analysis of the parts of graph G by Dijkstra algorithm, but is not sufficient to find the optimal route. We must combine found sub-routes into routes and then find the best route according to the chosen criteria. Moreover, the parallel Dijkstra algorithm can also be used for both the whole or each part of the graph. Firstly, the main loop can be parallelized. As for the parallelization of nested loops, it is not so obvious because we should synchronize calculations, which is time-consuming.

The organization of different simulations requires a suitable computing environment to test the proposed solutions. Therefore, on the one hand, we use a memory multicore architecture to speed up SAA execution; on the other hand, we choose a cloud environment to test the proposed solution.

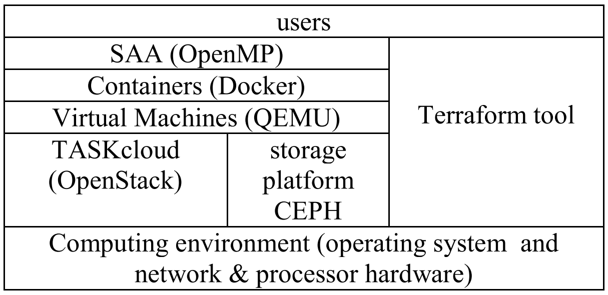


FIGURE 7. The simulation environment.

This environment can be described using a multilayer model (see Fig. 7). It comprises six main layers that perform different functions. The lowest layer is a basic computing environment, where an operating system (Ubuntu software) is chosen and installed on available computer hardware. The second layer represents the cloud environment and storage platform. It was created based on the Infrastructure as a Service (IaaS) model available in the cloud named TASKcloud. OpenStack [33] software was used to provide services to create virtual machines, block storage, and network components using an application programming interface (API). The next two layers create a virtual environment supported by QEMU and containers based on Docker [34]. The last layer corresponds to the SAA application implemented in OpenMP and is distributed as Docker containers. Container encapsulation of SAA software allows the inclusion of all

SAA dependencies, source code, runtime execution environment, system tools, and libraries inside containers. The upper layer represents SAA application users who can run it sequentially or in parallel. The SAA interface plays an essential role in user-to-cloud communication.

To facilitate the implementation of the entire system and achieve repeatability of simulations, the popular Terraform [40] tool was used. Terraform-dedicated scripts were developed to prepare the following infrastructure: virtual machine with 24 virtual threads, 60 GB of RAM, and 240 GB SSD. Additionally, the virtual machine attached a 100 GB extra disk with triple replication for safe data storage. The SAA virtual machine was attached to the Internet through a router and was assigned a public IP for users to access it. Terraform scripts allow a researcher to create or recreate the simulation environment, repeatable and within a few minutes.

The migration of SAA software to the cloud requires the following three steps: (1) preparing the virtual infrastructure, (2) configuring the operating systems, and (3) deploying the SAA software itself. The proposed solution allows us to run the SAA application for customized regional resolution, frequency of weather forecasting data, and selected scenarios. The required data can be collected by either dedicated scripts or a cloud service; the latter is more convenient and can be performed automatically. All gathered data has been stored on cloud CEPH due to safety reasons. In addition, we compared the runtimes of SAA in both traditional (sequential) and cloud (parallel) environments for various user requests. The parallelized SAA application has been containerized in a single Docker container and has been run on a single virtual machine. This approach lets us scale SAA application horizontally and run a dedicated containerized instance per each new route planning request.

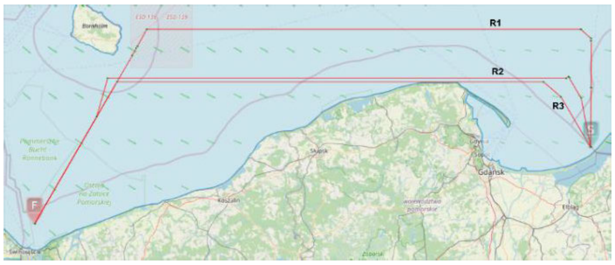


FIGURE 8. SSA routes example. Route (R1,R2,R3) is based graph includes (14523121, 3630780, 907695) number of vertices and (2.53E+08, 63058011, 7840569) number of edges.

V. A COMPARISON OF PARALLEL AND SEQUENTIAL VERSIONS OF SAA

To analyse the benefits of the proposed SAA parallelization, we performed several simulations in a prepared cloud environment. First, we considered the Baltic region described by different grid resolutions. Following this, we created graphs, considered different sailing conditions, and modelled sailing times as weights assigned to graph edges. Finally, we ran the SAA (Fig. 8 shows some sample routes displayed in this application) and measured the processing times of SSA components for different input data, as shown in Table 1 and Fig 9.

TABLE 1. Comparison of execution times for sequential (Seq) and parallel (Prl) processing of SAA.

Parameters of graph (navigation area)			Q Execution time of SAA component [in second]			Reduction in execution time due to the used parallelization	
Number of vertices	Vertex degree	Number of edges	Generating navigational area	Establishing sailing conditions	Finding a route	Total time [sec]	%
907695	8	3926310	Seq:0.19 Prl:0.13	Seq:1.21 Prl:0.58	Seq: 0.21	Seq: 1.61 Prl:0.92	43
907695	16	7840569	Seq:0.19 Prl:0.13	Seq:1.65 Prl:0.86	Seq: 0.9	Seq: 2.74 Prl:1.89	31
907695	32	15645055	Seq:0.19 Prl:0.13	Seq:2.98	Seq: 2.10	Seq: 5.27	32
				Prl:1.34		Prl:3.57	
3630780	8	15799229	Seq:0.24 Prl:0.19	Seq:3.4 Prl:1.78	Seq: 0.54	Seq: 4.18 Prl:2.51	40
3630780	16	31571011	Seq:0.24 Prl:0.19	Seq:6.92 Prl:3.37	Seq: 2.26	Seq: 9.42 Prl:5.82	38
3630780	32	63058011	Seq:0.24 Prl:0.19	Seq:15.16 Prl:6.28	Seq: 7.67	Seq: 23.07 Prl:14.14	39
14523121	8	63421793	Seq:0.63 Prl:0.39	Seq:15.87 Prl:5.98	Seq: 2.44	Seq: 18.94 Prl:8.81	53
14523121	16	126784091	Seq:0.63 Prl:0.39	Seq:24.41 Prl:10.83	Seq: 6.48	Seq: 31.52 Prl:17.7	44
14523121	32	253382867	Seq:0.63 Prl:0.39	Seq:52.25 Prl:19.58	Seq: 22.97	Seq: 75.85 Prl:42.94	43

The effect of parallelization is presented in the last column of Table 1. The growth of this effect as a function of the number of graph-edge parameters is shown in Fig. 9. An example of an SAA interface is shown in Fig. 8. This illustrates three different routes along the Polish coast.

VI. DISCUSSION OF RESULTS

It can be observed that utilized parallelization is beneficial for the first two components of SAA application. For the last component of the SAA, the parallel approach failed to reduce

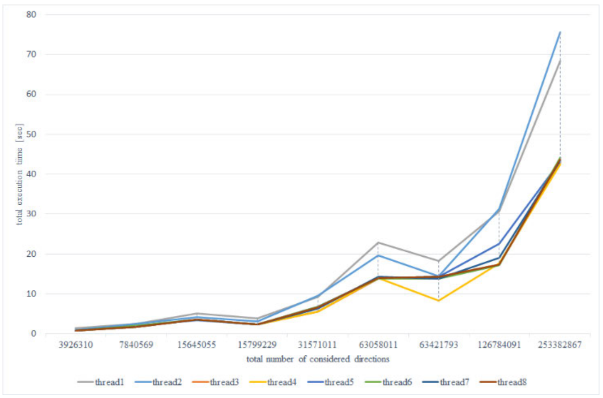


FIGURE 9. Total execution time as a function of the number of edges in the graph for various numbers of parallel threads.

the processing time. For this component, we obtained much larger execution times for the parallel version of SAA than for the sequential version. Therefore, we created a new SAA configuration: parallel versions of two components (generating navigational area and establishing sailing conditions) and a sequential version of the last component (finding a route), which provides a mean effect of speed-up equal to 40.4%.

The main reason for the inefficiency of the third component is that the applied structure of the data – the heap – requires many synchronizations, thus resulting in longer computations. It is worth noting that for the second SAA task (establishing sailing conditions), the parallelized approach is more effective if the number of possible routes between the endpoints is higher. We also analysed the total time as a function of the number of threads supporting the parallelization. This relation corresponds to the speed-up of parallel applications; speed-up is a measure that captures the relative benefit of solving a problem in parallel. In our case, it is defined as the ratio of the time taken to solve a problem using the sequential algorithm to the time required to solve the same problem in a shared-memory multicore architecture. If the speed-up is linear regarding the number of multithread, the assumed solution can significantly shorten the required time. However, for many problems, such scalability is difficult to achieve owing to synchronization and communication problems. As shown in Fig. 6, this is also true for the considered problem. Parallelization reduces the computational time for up to four engaged threads. Further increase in the number of threads did not result in additional benefits. This issue will be investigated in ongoing research on the presented solution. Alternative parallel approaches to the final SAA task can be considered. It must be mentioned, however, that Dijkstra's algorithm has not only high computational time, but also high computational memory. It is also not scalable without a loss in accuracy. These two features render parallelization particularly difficult.

VII. CONCLUSION

Summing up the presented research, we can state the following.

The original sequential application SAA was transformed into a parallel one, where data parallelism was considered. We have uses open software to implement parallel SAA (OpenMP) and to create a computing environment (TASKcloud) to test created application. In consequence, we obtain a practically oriented solution.

We improve performance of two parallel ASS components by up to 79.1% and show that the selected number of threads is optimal in terms of minimizing execution time. In general, it depends on the size of navigation area, resolution, and the number of divided part of data, which is assigned to all threats. To find the optimal route, we use the sequential Dijkstra algorithm. While there are some parallel Dijkstra algorithms, the majority of them consider an input data set described by a matrix and are tested rather for small data sizes (hundreds of elements instead of a million). In our case, more complex optimization criteria (e.g. penalty for manoeuvres) and some dynamic parameters (e.g. weather forecasting) have been applied and the parallel Dijkstra version did not fulfil our expectations in terms of flexibility of problem modelling and implementation. Choice of more advanced data structures, such as Fibonacci heap, can lead to better results.

Another potential direction of research is investigating other models of parallelism – MPI and MPI/OpenMP, especially for determining simultaneously different routes for different ships in the same navigation region. In general, due to its flexible, module-based structure, the parallel SAA application is open to various changes in terms of optimization and its parallel implementations. Therefore, all the above-mentioned aspects will be explored in the ongoing work on the proposed method.

REFERENCES

[1] M. F. Maury, *The Physical Geography of the Sea and Its Meteorology*. New York, NY, USA: Harper Bros, 1855.

[2] *SOLAS—International Convention for the Safety of Life at Sea*, International Maritime Organization, London, U.K, 2009.

[3] M. Życzkowski, J. Szłapczyńska, and R. Szłapczyński, "Review of weather forecast services for ship routing purposes," *Polish Maritime Res.*, vol. 26, no. 4, pp. 80–89, Dec. 2019.

[4] M. Życzkowski and R. Szłapczyński, "Multi-objective weather routing of sailing vessels," *Polish Maritime Res.*, vol. 24, no. 4, pp. 10–17, Dec. 2017.

[5] P. Orzechowski, "Task cloud infrastructure in the centre of informatics—Tricity academic supercomputer & network," *Task Quart.*, vol. 22, no. 4, pp. 313–319, 2018.

[6] Y. Huang, L. Chen, and P. H. A. J. M. van Gelder, "Generalized Velocity Obstacle algorithm for preventing ship collisions at sea," *Ocean Eng.*, vol. 173, pp. 142–156, Feb. 2019.

[7] X. Liu, Y. Li, J. Zhang, J. Zheng, and C. Yang, "Self-adaptive dynamic obstacle avoidance and path planning for USV under complex maritime environment," *IEEE Access*, vol. 7, pp. 114945–114954, 2019.

[8] A. Morge, V. Pelle, J. Wan, and L. Jaulin, "Experimental studies of autonomous sailing with a radio controlled sailboat," *IEEE Access*, vol. 10, pp. 134164–134171, 2022.

[9] A. Zak, "Control of unmanned underwater vehicle as a member of vehicles team performing a given task," *Trans. Maritime Sci.*, vol. 8, no. 1, pp. 18–25, Apr. 2019.

[10] J. Szłapczyńska, R. Vettor, R. Szłapczyński, M. Łacki, M. Życzkowski, M. A. Hinostroza, F. P. Santos, W. Tycholiz, and C. G. Soares, "Weather routing system architecture using onboard data collection and route optimisation," *Polish Maritime Res.*, vol. 29, no. 2, pp. 87–95, Jun. 2022.

[11] V. Lehtola, J. Montewka, F. Goerlandt, R. Guinness, and M. Lensu, "Finding safe and efficient shipping routes in ice-covered waters: A framework and a model," *Cold Regions Sci. Technol.*, vol. 165, Sep. 2019, Art. no. 102795.

[12] T. Zvyagina and P. Zvyagin, "A model of multi-objective route optimization for a vessel in drifting ice," *Rel. Eng. Syst. Saf.*, vol. 218, Feb. 2022, Art. no. 108147.

[13] M. Życzkowski, P. Krata, and R. Szłapczyński, "Multi-objective weather routing of sailboats considering wave resistance," *Polish Maritime Res.*, vol. 25, no. 1, pp. 4–12, Mar. 2018.

[14] A. B. Philpott, I. M. Viola, and R. G. J. Flay, "Optimal yacht routing tactics," in *Proc. 3rd Int. Conf. Innov. High Perform. Sailing Yachts*, 2013, pp. 231–237.

[15] A. Philpott and A. J. Mason, "Optimising yacht routes under uncertainty," Andy Philpott, Yacht Res. Unit, Univ. Auckland, New Zealand, Oceania, Tech. Rep., 2014.

[16] M. Życzkowski, "Sailing vessel routing considering safety zone and penalty time for altering course," *TransNav, Int. J. Mar. Navigat. Saf. Sea Transp.*, vol. 11, no. 2, pp. 49–54, 2017.

[17] M. Życzkowski and R. Szłapczyński, "Collision risk-informed weather routing for sailboats," *Rel. Eng. Syst. Saf.*, vol. 232, Apr. 2023, Art. no. 109015.

[18] J. Szłapczyńska and R. Szłapczyński, "Preference-based evolutionary multi-objective optimization in ship weather routing," *Appl. Soft Comput.*, vol. 84, Nov. 2019, Art. no. 105742.

[19] H.-B. Wang, X.-G. Li, P.-F. Li, E. I. Veremey, and M. V. Sotnikova, "Application of real-coded genetic algorithm in ship weather routing," *J. Navigat.*, vol. 71, no. 4, pp. 989–1010, Jul. 2018.

[20] E. Alhenawi, R. A. Khurma, A. A. Sharieh, O. Al-Adwan, A. A. Shorman, and F. Shannaq, "Parallel ant colony optimization algorithm for finding the shortest path for mountain climbing," *IEEE Access*, vol. 11, pp. 6185–6196, 2023.

[21] S. A. M. Ali and E. H. Al-Hemiary, "A parallel implementation method for solving the shortest path problem for vehicular networks," in *Proc. 1st. Inf. Technol. Enhance E-Learn. Appl. (IT-ELA)*, Jul. 2020, pp. 121–126.

[22] Z. Cheng, L. Zhao, and Z. Shi, "Decentralized multi-UAV path planning based on two-layer coordinative framework for formation rendezvous," *IEEE Access*, vol. 10, pp. 45695–45708, 2022.

[23] N. Edmonds, A. Breuer, D. Gregor, and A. Lumsdaine, "The shortest path problem," *Single-Source Shortest Paths With the Parallel Boost Graph Library*. 2009, pp. 219–248.

[24] J. Kim, "HGED: A hybrid search algorithm for efficient parallel graph edit distance computation," *IEEE Access*, vol. 8, pp. 175776–175787, 2020.

[25] D. Josifoski, M. Gusev, V. Zdraveski, and S. Ristov, "Parallelization of Dijkstra's algorithm for robot motion planning: Is it worth increasing speed without losing accuracy?" in *Proc. 30th Telecommun. Forum (TELFOR)*, Nov. 2022, pp. 1–4.

[26] J. H. Seo, H. Lee, and K.-D. Kim, "A parallelization algorithm for real-time path shortening of high-DOFs manipulator," *IEEE Access*, vol. 9, pp. 123727–123741, 2021.

[27] M. He, "Parallelizing Dijkstra's algorithm," St. Cloud State Univ., St. Cloud, MN, USA, Tech. Rep., 2021.

[28] R. van der Pas, E. Stotzer, and C. Terboven, *Using OpenMP—The Next Step: Affinity, Accelerators, Tasking, and SIMD*. Cambridge, MA, USA: MIT Press, 2017.

[29] P. Czarnul, J. Proficz, and K. Drypczewski, "Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems," *Sci. Program.*, vol. 2020, Jan. 2020, Art. no. 4176794.

[30] H. Krawczyk, M. Nykiel, and J. Proficz, "Tryton supercomputer capabilities for analysis of massive data streams," *Polish Maritime Res.*, vol. 22, no. 3, pp. 99–104, Sep. 2015.

[31] S. Manvi and G. Shyam, *Cloud Computing: Concepts and Technologies*. 2021.

[32] J. Grundy, G. Kaefer, J. Keung, and A. Liu, "Guest editors' introduction: Software engineering for the cloud," *IEEE Softw.*, vol. 29, no. 2, pp. 26–29, Mar. 2012.

[33] S. Pal, D.-N. Le, and P. K. Pattnaik, *Cloud Computing Solutions? Architecture, Data Storage, Implementation, and Security*. 2022.

[34] D. E. Comer, *The Cloud Computing Book; The Future of Computing Explained*. 2021.

[35] A. H. Day, "Performance prediction for sailing dinghies," *Ocean Eng.*, vol. 136, pp. 67–79, May 2017.

[36] M. Życzkowski, "Sailing route planning method considering various user categories," *Polish Maritime Res.*, vol. 27, no. 3, pp. 149–158, Sep. 2020.

[37] W. Fan, K. He, Q. Li, and Y. Wang, "Graph algorithms: Parallelization and scalability," *Sci. China Inf. Sci.*, vol. 63, no. 10, pp. 1–21, Oct. 2020.

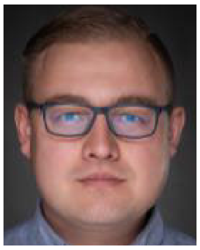
[38] W. Kocay and D. L. Kreher, *Graphs, Algorithms, and Optimization*, 2nd ed. Nov. 2016, pp. 1–546.

[39] R. Baños, J. Ortega, C. Gil, F. de Toro, and M. G. Montoya, "Analysis of OpenMP and MPI implementations of meta-heuristics for vehicle routing problems," *Appl. Soft Comput.*, vol. 43, pp. 262–275, Jun. 2016.

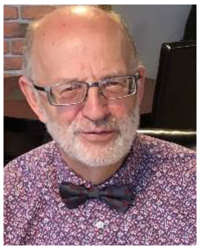
[40] R. Mishra and O. M. C. Safari, *HashiCorp Infrastructure Automation Certification Guide*. Birmingham, U.K.: Packt Publishing, 2021.

MARCIN ZYCZKOWSKI received the Ph.D. degree in transport from Gdynia Maritime University, in 2018. He is currently an Assistant Professor with the Faculty of Mechanical Engineering and Ship Technology, Gdańsk University of Technology. His current research interests include deterministic algorithms, big data, and parallel programming applied in the maritime transport applications. The preferred programming language is python.

RAFAL SZLAPCZYNSKI received the D.Sc. degree in transport from the Warsaw University of Technology, in 2015. He is currently an Associate Professor and the Head of the Computer Science Department, Faculty of Mechanical Engineering and Ship Technology, Gdańsk University of Technology. His current research interests include soft computing, especially multi-objective meta heuristics and their applications in marine navigation and transport.



PIOTR ORZECHOWSKI received the M.Sc. degree in computer science from the Gdańsk University of Technology, in 2011. He is currently the Head of the IT Services and Applications Department, Centre of Informatics Tricity Academic Supercomputer and Network, Gdańsk University of Technology. His current research interests include cloud computing, workload scheduling, resource management in cloud infrastructure, and automation of deployment processes.



HENRYK KRAWCZYK is currently with the Faculty of Electronics, Telecommunications and Informatics and the TASK Informatics Center, Gdańsk University of Technology. He is also the Director of the Competence Center for Smart Transdisciplinary Knowledge Services. His current research interests include high-performance computing, service-oriented software engineering, and dependable distributed systems.