Dynamic Computational Resource Allocation for CFD Simulations Based on Pareto Front Optimization

Gašper Petelin
Jožef Stefan Institute
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia
gasper.petelin@ijs.si

Margarita Antoniou
Jožef Stefan Institute
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia
margarita.antoniou@ijs.si

Gregor Papa
Jožef Stefan Institute
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia
gregor.papa@ijs.si

ABSTRACT

Computational Fluid Dynamics (CFD) simulations can be extremely computationally demanding and usually rely on the use of High-performance computing (HPC) using both CPU and GPU resources. Modeling the behavior of bubbles size distribution often leads to a symmetric function evaluation problem. This paper proposes a dynamic computational resource allocation, based on Pareto-optimal solutions. The solutions are obtained from the formulation of the resource-constrained symmetric function evaluation problem as a multi-objective problem. After the Pareto-front is obtained, we suggest a dynamic selection method of the solutions that utilize the existing resources. To solve the multi-objective problem ϵ-MOEA, an algorithm known to obtain good diversity of Pareto-front solutions, is applied. As the problem formulated is new, brute-force search and two-specifically designed for this problem-heuristics are implemented and tested to serve as baselines. The methods are tested and compared to three dimensionalities of the problem. The results showed that ϵ-MOEA can successfully approximate the Pareto-front, allowing to utilize the resources optimally at each simulation time-step.

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**; • **Applied computing** → *Operations research.*

KEYWORDS

Resource Allocation, Multi-objective Optimization, Computational Fluid Dynamics, High-Performance Computing

1 INTRODUCTION

A challenge that is frequently encountered in Computational Fluid Dynamics (CFD) applications is how to accurately model the behavior of particles (i.e., bubbles, powder) [9, 15]. Such models are especially relevant for nuclear and process engineering where one is interested in the behavior of bubbles of different sizes (bubble size distribution). Bubble size distribution can be tracked with population balance equation [21], which incorporates the effects of coalescence (particles agglomerate into bigger ones) and breakup (particles split into smaller ones). The method of classes [13] models the distribution of bubbles by discretizing the bubble population into a finite number of size groups (visualised in Figure 1). All of the interactions between bubble size groups are then aggregated by source term assembly and used to produce a new updated distribution. Assembling the source terms involves the computation of coalescence and breakup frequencies between all bubble-size pairs.

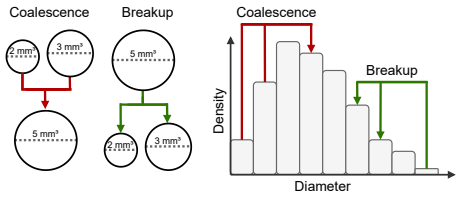

Figure 1: Bubble size distribution is discretized into nine groups that are tracked for each cell in the mesh and mechanisms of coalescence and breakup that change the distribution during the simulation.

Therefore, one comes across the following task: to evaluate a symmetric function of different inputs at each time step of the simulation. With finer meshes and a large number of size groups, this can become one of the most computationally expensive parts of the simulation [19].

The naive approach to evaluating the symmetric function is to calculate each pair sequentially. This leads to a large waste of resources, as it requires recomputing the same values and takes no optimal advantage of the memory that is available in the HPC system. In addition, one should take into account the different associated CPU and GPU costs for each computation of the pairs.

Finding the optimal order of pair inputs in the symmetric functions leads to a constrained combinatorial problem. Taking also into account the objectives mentioned above makes the problem multi-objective. Multi-objective Evolutionary Algorithms (MOEAs) have become popular in solving hard combinatorial problems, such as the MOEA/D for the flowshop scheduling problem [5] and MOEA/D and NSGAII on the multi-objective traveling salesman problem [18]. Most multi-objective evolutionary algorithms are based on Pareto dominance and it is the measure of the quality of their solutions during the search. Using the MOEAs, one can obtain a near-optimal set of non-dominated solutions, giving the freedom of choice between conflicting objectives.

Efficiently allocating the available resources is of great interest for CFD and it can be done based on the different constraints (user requests, resource types, etc.) as in [1, 4]. Increasing the utilization

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

of the available resources with the HPC systems can be dramatically improved by the scheduling of jobs as shown in [11, 14, 22]. A dynamic resource allocation for efficient parallel CFD simulations that adapt at run-time is proposed in [10]. In CFD simulations parallelization is often already implicitly handled by the frameworks [20] that (during the simulation) splits the mesh into multiple regions and process them independently in parallel. Such is the case also in our scenario where a level of parallelization is already integrated into the framework. Therefore, we are interested in a dynamic computational resource allocation without the need to manually manage the parallelization.

This paper tackles the following research questions:

(1) Taking into account the different associated costs and the memory constraints of the HPC system, which is the optimal order of computing each pair of the symmetric function?
(2) Since we have two objectives - minimizing the CPU and GPU costs - leading to a multi-objective problem, is it possible to obtain a number of equally good solutions, that form a Pareto-front, with multi-objective evolutionary algorithms?
(3) If this is the case, why not use at each time-step of the simulation (that needs the calculation of the same symmetric function with different inputs) the most adequate ordering of the CPU-GPU Pareto-front to further, dynamically, utilize the available resources?

As the problem is new, brute-force search and two specifically designed problem heuristics are also implemented and tested, to serve as baseline solutions. After the Pareto-front is obtained, a dynamic selection method of the solutions is proposed aiming to utilize the existing resources. The methods are tested and compared to three dimensionalities of the problem. To find the approximation set (i.e., the closest solution to the theoretical Pareto-front) of the multi-objective problem, we applied five MOEAs (MOEA/D [23], NSGAII [6], GDE3 [12], ϵ-MOEA [7], SPEA2 [24]). The ϵ-MOEA showed superior performance in terms of fast convergence, and the reported results of the paper refer to this algorithm. The empirical comparison of the MOEAs is out of the scope of this paper (some more results are available in the Appendix A.

We organize the rest of this paper as follows. In Section 2 the problems of dynamic resource allocation and the resource-constrained symmetric function evaluation are formulated and described. The methods developed and used to solve the problem are shortly described in Section 3. The experimental setup and the obtained results are discussed in Section 4. The last section summarizes the conclusions of the paper, as well as some potential research directions.

2 PROBLEM FORMULATION

In this section, we formulate the dynamic resource allocation problem, which contains the resource-constrained symmetric function evaluation problem.

2.1 Dynamic Allocation of Computational Resources

In each time step of the simulation, different CPU and GPU resources are available. At the same time, each tensor/input consumes GPU and CPU resources. The goal is to dynamically utilize the resources during the different time-steps. Based on the number of the size groups, the available memory, and the cost of the inputs, the source term assembly represents a symmetric function that has to be evaluated on all pairs of bubble sizes. The associated challenge is to determine the order in which to assemble source terms and the order of recomputing tensors to minimize resource use. Obtained solutions for assembling source terms can therefore differ in the amount of CPU and GPU resources that they consume. A challenge, therefore, becomes how to select in each time-step of the simulation what order should be used to assemble source terms based on the current utilization of available resources.

2.2 Resource-Constrained Symmetric Function Evaluation

Here, the resource-constrained symmetric function evaluation problem, with its mathematical formulation and notation, is described.

Problem $P_{n,l}$ is defined as follows: assume we have a set of n inputs $\mathbf{I} = \{I_0, I_1, \ldots, I_{n-1}\}$ and a function $f(I_j, I_k)$ that has to be evaluated for all the input pairs. The function is symmetric, thus $f(I_j, I_k) = f(I_k, I_j)$. For a memory of size l (where $l < n$), one can store a subset of l inputs at a given time. When the function $f(I_j, I_k)$ is being evaluated with new input, one previously obtained input is dropped from the memory. Then, the new input is calculated and stored in the memory, before evaluating the function. We are interested in finding the optimal order of the subset of inputs to be evaluated from the function, that minimizes their re-computations.

Moreover, each input I_k has one or more costs associated with its computation, creating an additional constraint to the problem. Let the $r_i(I_k) > 0$ be a function that describes the amount of type i resource needed for calculating the input I_k. *The goal is to find a set of solutions (i.e., orders) that will guarantee all pairs to be evaluated while a) never exceeding the memory constraints and b) minimizing re-computation costs.*

Let O be one possible ordering of the inputs. Each objective value is defined as $obj_i = \sum_{I_k \in O} r_i(I_k)$. The goal is to find a set of orderings that minimize the objectives.

Figure 2 shows a problem $P_{5,3}$ with randomly selected costs and one possible solution to the problem. The set of inputs is of size 5 ($n = 5$) and the memory size is 3 ($l = 3$). One possible solution is the order $[I_0, I_1, I_2, I_3, I_4, I_0, I_1]$. For this ordering, the objective values are $(4.4, 4.2)$. Alternatively, the ordering of $[I_0, I_2, I_3, I_4, I_0, I_1, I_3, I_2]$ has objective values of $(5.5, 4.6)$ and is thus inferior to the first ordering. In general, depending on the costs associated with each input I_k in the set, the problem can have one or multiple optimal solutions (i.e., solutions on the Pareto front).

3 METHODS OVERVIEW

3.1 Brute-force Search

The problem is new with no known optimal solutions. To obtain the solutions, one can check all the possible candidate solutions, known as a brute-force search. We apply brute force to explore the search space and obtain baseline/reference solutions. This approach is only possible for low dimensions of the problem, as it becomes prohibitively costly - or even infeasible - as the dimensions increase.

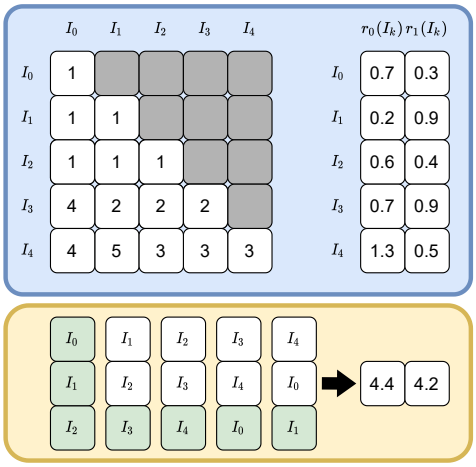

Figure 2: Example of a problem $P_{5,3}$ with randomly selected costs for all the inputs (top). One possible solution (bottom) is described with objective values of (4.4, 4.2). The first triplet in memory is (I_0, I_1, I_2) ensuring that first 6 function evaluations will be $[f(I_0, I_0), f(I_0, I_1), f(I_1, I_1), f(I_0, I_2), f(I_1, I_2), f(I_2, I_2)]$. Inputs marked with green are computed and placed into the memory while white ones are just reused.

3.2 Deterministic Heuristics

To further investigate the solutions, we propose two heuristic non-evolutionary algorithms for computing the ordering of inputs. Such heuristics serve as a fast and efficient baseline used to evaluate more complex evolutionary algorithms. Heuristic algorithms only produce one final solution contrary to MOEAs. Our two proposed deterministic heuristics work in the following way:

- *Heuristic 1* - It first computes the inputs that are determined as the most expensive to evaluate (*select_expensive_inputs*) and stores them in the available memory. Because the inputs have a CPU and a GPU cost associated with them, the most computationally expensive inputs are determined based on the sum of the objectives (i.e. sum of both CPU and GPU resources they consume). After the memory is filled with expensive-to-evaluate inputs and the cost of computing them is added to the total cost, we iterate over all the input pairs that are required to evaluate the function. The order in which the input pairs are iterated over is determined based on how many of the inputs in the input pair they share memory with. Pairs, where both of the inputs are already in memory, are processed first followed by pairs where one input is already in memory, and lastly pairs where no input is in memory. Order is therefore determined with method *select_best_pair*;
- *Heuristic 2* - It is similar to the first one with one significant modification. During the execution, when a certain input is no longer needed, it is dropped from the memory and replaced with the inputs that were just calculated (*update_memory*). Updating of the memory is thus performed as follows: When a new input that is not in stored the memory is calculated, a check is made to determine if any of the inputs in memory is no longer needed (due to all the input pairs that rely on that input already being calculated). If this is the case, the input from the memory is replaced with the one that was just calculated.

Algorithm pseudocode 1 describes how both algorithms work. The only difference between *Heuristic 1* and *Heuristic 2* is that *Heuristic 2* updates the inputs in memory as described in line 20 (i.e., *Heuristic 2* only differs from *Heuristic 1* in additional line 20).

Algorithm 1 Algorithm pseudocode for *Heuristic 1* and *Heuristic 2*. Algorithms differ only in how memory is updated in line 20. Line 20 is only used in *Heuristic 2*

1: $cost_{CPU} \leftarrow 0$
2: $cost_{GPU} \leftarrow 0$
3: $p \leftarrow generate\_list\_of\_input\_pairs(inputs)$
4: $m \leftarrow select\_expensive\_inputs(inputs)$
5: **for** $Input \in m$ **do**
6: $\quad cost_{CPU} \leftarrow cost_{CPU} + r_{CPU}(Input)$
7: $\quad cost_{GPU} \leftarrow cost_{GPU} + r_{GPU}(Input)$
8: **end for**
9: **while** $len(p)! = 0$ **do**
10: $\quad c \leftarrow select\_best\_pair(m, p)$
11: $\quad remove\_pair\_from\_list(c, p)$
12: $\quad$ **if** $c.Input_1 \notin m$ **then**
13: $\quad\quad cost_{CPU} \leftarrow cost_{CPU} + r_{CPU}(c.Input_1)$
14: $\quad\quad cost_{GPU} \leftarrow cost_{GPU} + r_{GPU}(c.Input_1)$
15: $\quad$ **end if**
16: $\quad$ **if** $c.Input_2 \notin m$ **then**
17: $\quad\quad cost_{CPU} \leftarrow cost_{CPU} + r_{CPU}(c.Input_2)$
18: $\quad\quad cost_{GPU} \leftarrow cost_{GPU} + r_{GPU}(c.Input_2)$
19: $\quad$ **end if**
20: $\quad m \leftarrow update\_memory(m, p, c)$
21: **end while**
22: **return** $(cost_{CPU}, cost_{GPU})$

3.3 ϵ-MOEA

The ϵ-MOEA proposed by Deb et. al. [7] is a steady-state algorithm, incorporating ϵ- dominance archiving. The algorithm generates only one solution in each iteration. It consists of two co-evolving populations, the EA population-which starts as random- and an archive population-which gets the non-dominated solutions the former produced. Using the ϵ-dominance concept, in each generation, a new offspring is produced by two solutions of the two populations. The ϵ-dominance enables the algorithm to approximate the Pareto front well both in terms of convergence and diversity. More details can be found in [7].

3.3.1 Encoding. The representation of an individual consists of one chromosome. Each chromosome encodes the order in which inputs have to be dropped or recomputed as a list of inputs represented as integers. Since it is possible that the inputs have to be recomputed, they can occur on the list multiple times. A number of occurrences of a certain input in the list tell us how many times it has to be recomputed.

3.3.2 Initialization. The construction of the initial population can have a significant impact on the optimization procedure [2, 8]. We initialised the chromosome as a random ordering of inputs.

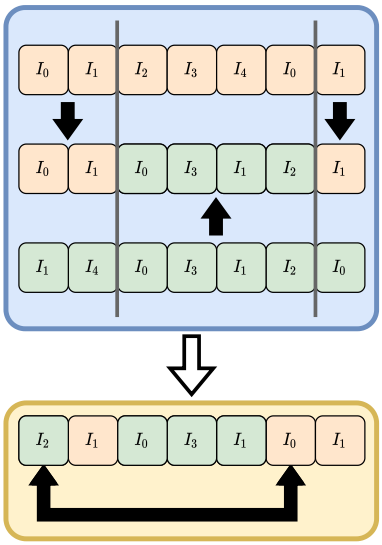

Figure 3: Example of a two-point crossover (top) where the offspring chromosome is created based on two randomly selected parents. The offspring is then mutated (bottom),where two genes (inputs) are swapped.

3.3.3 Crossover and mutation operators. During the optimization procedure, new chromosomes are generated with the goal of improving the orderings to minimize costs. To achieve this, EAs apply the crossover and mutation operators. The crossover operator transfers information from parents to their offspring. Generally, when dealing with permutations (i.e. traveling salesman problem) there exists a wide range of crossover operations where the goal is to never produce a chromosome that is not a valid permutation. In our case the ordering is not a permutation, as repeated values are allowed. We are thus able to use different crossover operators. We used a two-point crossover where two-parent chromosomes are sliced and everything between the two points is swapped. After performing crossover, mutation is also applied. We used swap mutation where order of two inputs are swapped to obtain a new chromosome. Figure 3 shows an example of both the crossover and mutation operators that are used during the optimization.

3.4 Dynamic Selection Method

To solve the dynamic resource allocation problem, one has to select an adequate solution from the multiple non-dominated solutions presented in the approximation set Pareto-front. In our case, the two objective functions are the GPU and CPU resources consumed during the computation. At each time step of the simulation, different GPU and CPU resources are available. Our goal is to select the solution that takes advantage better the available resources.

Numerous approaches exist on how to select a solution from the Pareto-front. If the ideal point is known, one can rank the obtained solution and select the best one such as in [16]. If the ideal solution is not known, a strategy of finding a "knee" in the obtained solutions can be used as described in [3].

In our problem, we applied the following method, which is fast and does not require previous knowledge about the Pareto-front form:

- At the beginning of each time step of the simulation, we check the utilization of the resources (GPU and CPU) on the system where simulation is running.
- Based on their utilization, we calculate the following ratio:

$$ratio_t = \frac{util^t_{CPU}}{util^t_{GPU} + \epsilon} \tag{1}$$

  where $util^t_{CPU}$ and $util^t_{GPU}$ are utilizations of both resources at time step t and ϵ is some small constant (in our case 0.001) to prevent division by zero if GPU is in idle state.
- Next, we use the ratio as a slope of the line that goes through the point (GPU_{min}, CPU_{min}), where the GPU_{min} and CPU_{min} are the minimum values of those two objectives that are obtained from some solution. We need to point here that their selection is independently done for each time step of the simulation.
- To decide which solution will be further selected, we calculate the distance of each non-dominated solution from the approximation set and the line. The solution that is closer to the line is selected as the most relevant one. The selected solution, therefore, represents an ordering that describes a trade-off between CPU and GPU resource usage according to the resource utilization for each time step of the simulation.

An example of the aforementioned framework can be seen in Figure 4.

Step 1: Based on the number of size groups, available memory, and the cost for inputs, we formulate the optimization problem.
Step 2: we obtain the Pareto front of our multi-objective problem.
Step 3: for each time step of the simulation, the most optimal solution is selected according to the ratio above.

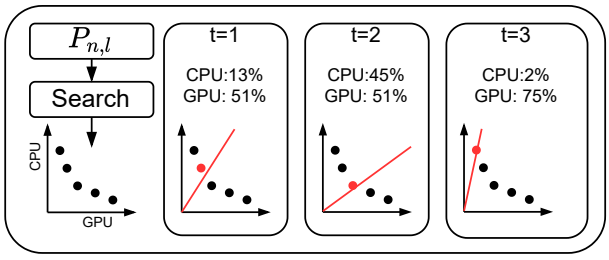

Figure 4: Obtaining a set of Pareto optimal solutions at the beginning of the simulation and dynamically selecting the most optimal ordering for each time step during the process of the simulation based on the resource availability.

Table 1: Hyperparameter values for the ϵ-MOEA.

Hyperparameter	Value
Population size	50
Two-point crossover probability	0.5
Swap mutation probability	0.1
ϵ in ϵ-MOEA	0.1

4 RESULTS

4.1 Experimental Setup

For the ϵ-MOEA implementation, the platypus[1] Python framework is used. Workflow and analysis are performed using Snakemake [17] ensuring reproducibility. The instances are independently run 30 times on an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 1 TB of RAM, and the Ubuntu operating system. The relevant code can be found at the Gitlab repository[2]. In Table 1 we report the hyperparameter values used in the experiments. We kept the default hyperparameter values and we did not perform an extensive search for the best hyperparameter.

4.2 Test-problems

The ϵ-MOEA, the two heuristics, and the brute-force search were compared on a set of randomly generated instances, minimizing the objectives reported above. To test the scalability of the methods, we generated three instances of the problem, with increasing dimensionality. In specific, the three instances are of size $P_{5,3}$, $P_{15,3}$ and $P_{30,3}$. In CFD simulations, that would mean a bubble size distribution discretized into 5,15, and 30 groups respectively, and a memory that can store up to 3 tensors. The GPU and CPU costs were sampled randomly from an exponential distribution with mean=200 and a unique seed. Based on the CPU and GPU costs, the problem can have one or more solutions. How the generated costs affect the test functions and their subsequent solutions are outside of the scope of the paper.

4.3 Performance on Low-dimensional Instance

To better understand the behavior of the ϵ-MOEA and the heuristic algorithms on the problem, we evaluate them on the low-dimension instance, namely $P_{5,3}$. First, we obtain all the feasible solutions and the optimal Pareto front by the brute-force search. The non-dominated solutions found serve as the baseline solutions of the optimal Pareto-front for comparison with the other methods.

Figure 5 is a scatter plot of all the feasible solutions (grey dots) obtained by the brute-force search. The non-dominated solutions are the Pareto-front of the instance and are depicted with an empty circle. In the same plot, the solutions found by Heuristic 1 and Heuristic 2 are depicted with blue and red dots, respectively. The orange dots depict the non-dominated solutions found by the ϵ-MOEA after one run, which coincide with the solutions with the global optima. On the contrary, the solutions found by the heuristics are far from the known Pareto-front.

[1]https://platypus.readthedocs.io/en/latest/getting-started.html
[2]https://repo.ijs.si/gpetelin/square-resource-problems-2022

Table 2: Success rate of finding n non-dominated solutions out of 5 known solutions for problem $P_{5,3}$.

	Success Rate				
Number of solutions	1	2	3	4	5
Brute-force	1	1	1	1	1
Heuristic 1	0	0	0	0	0
Heuristic 2	0	0	0	0	0
ϵ-MOEA	1	1	1	0.866	0.566

In Table 2 we report the success rate of finding the set of the non-dominated solutions. The success rate is calculated as the average ratio of the number of runs where the algorithm finds n solutions of the 5 known solutions to the total number of runs. As seen previously in Figure 5, the unique solutions the heuristics provide are sub-optimal, therefore their success rate is zero. What is interesting is that the ϵ-MOEA can guarantee to find at least 3 different non-dominated solutions, while managing to find at least 4 and all 5 with percentages 86% and 56% respectively. This demonstrates that - for low dimensional cases - the ϵ-MOEA can discover successfully optimal solutions.

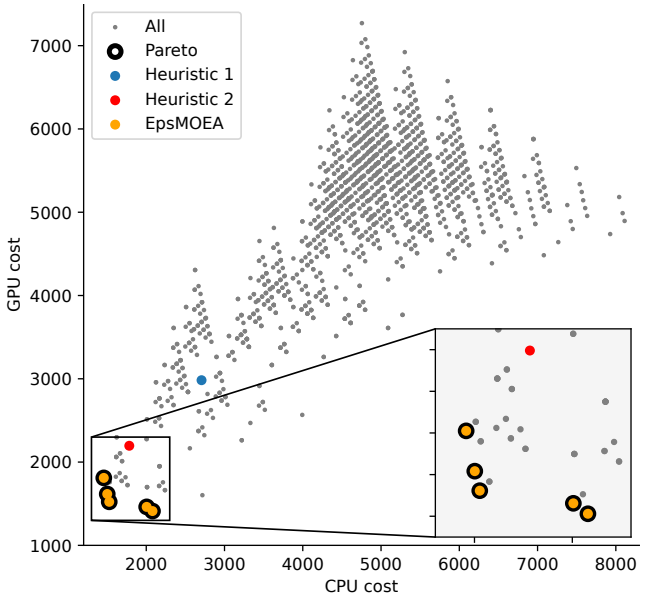

Figure 5: Scatter plot of all the feasible solutions for the problem $P_{5,3}$. The blue and red dots are the solutions obtained by Heuristic 1 and Heuristic 2, respectively. The orange dots belong to the Pareto front of one run of the ϵ-MOEA.

After the Pareto-front is obtained, the dynamic resource allocation is solved with the method explained in section 3. Figure 6 presents an example of how the solutions are dynamically selected (bottom), based on the CPU and GPU resource utilization (top) during seven simulation time steps. If the GPU and the CPU utilization are approximately equal, a solution that balances the usage of both resources will be selected in the specific time step. In other cases,

the solutions selected manage to optimally take advantage of the available resources.

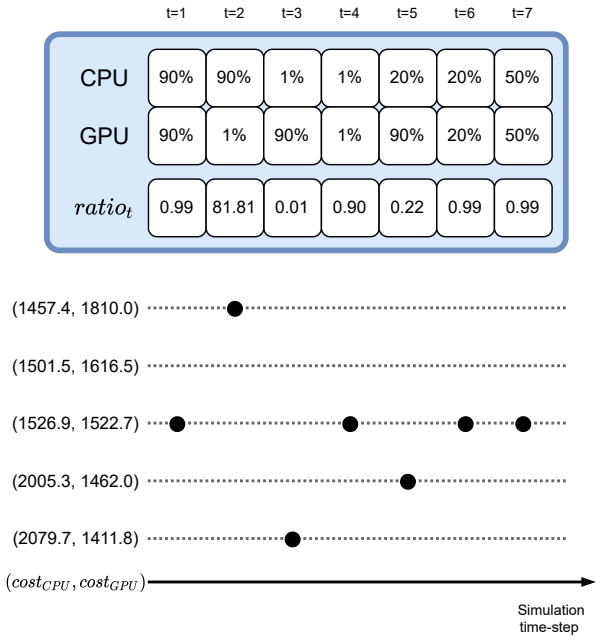

Figure 6: Example of how the solutions are dynamically selected (bottom), based on the CPU and GPU resource utilization (top) during 7 simulation time-steps.

4.4 Performance on Higher-dimensional Instance

To demonstrate the performance on more realistic problems, we selected two problems with sizes $P_{15,3}$ and $P_{30,3}$. Obtaining the solutions via brute force in these cases is not feasible. Therefore, for these problems, neither the optimal Pareto front nor the number of non-dominated solutions is known in advance.

In Table 3 we report the mean objective values for the two objectives, found by the two heuristics and the ϵ-MOEA. Regarding the developed heuristics, we can observe that *Heuristic 2* outperforms *Heuristic 1* in the instances. Dynamically selecting which inputs should be stored in memory, is thus beneficial compared to the static strategy of heuristic 1. The ϵ-MOEA finds solutions with significantly lower objective values, indicating that it is the most suitable method for this class of problems.

4.5 Convergence Speed

In this subsection, we focus on the convergence speed of the different methods, as it is important the most important aspect for the incorporation of the allocation in the simulation procedure.

Figure 7 presents the median convergence of the ϵ-MOEA in terms of a hypervolume for all the test instances over 30 runs. A reference point for the hypervolume computation was set as the worst solution, the one corresponding to computing sequentially all the pairs, without the possibility of storing them in the memory.

Table 3: Mean objective values of consumed CPU and GPU as obtained by Heuristic 1, Heuristic 2, and ϵ-MOEA. Values for the ϵ-MOEA are reported as a mean over 30 runs.

Problem	Algorithm	CPU resources	GPU resources
$P_{15,3}$	Heuristic 1	24086.7	53633.3
	Heuristic 2	13420.2	31984.3
	ϵ-MOEA	13421.6	17827.7
$P_{30,3}$	Heuristic 1	125055.1	167591.8
	Heuristic 2	61618.6	98636.1
	ϵ-MOEA	58412.6	56427.1

For $P_{5,3}$ and $P_{15,3}$, 50,000 function evaluations are enough to find good solutions, while instance $P_{30,3}$ requires approximately 100,000 function evaluations to reach convergence.

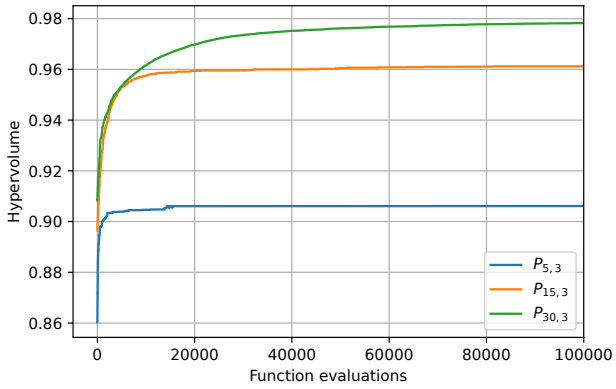

Figure 7: Median hypervolume indicator convergence of the ϵ-MOEA for problems $P_{5,3}$ (blue line), $P_{15,3}$(orange line) and $P_{30,3}$ (green line) over 30 runs.

Table 4 reports the mean runtime for the different tested methods. The runtime reported for the ϵ-MOEA is selected according to the observation above. Moreover, the brute-force method is only applied to the small instance, and thus no runtime is reported for the other two. For $P_{5,3}$, obtaining all the feasible solutions and the Pareto-front by brute-force search takes approximately 61s. The ϵ-MOEA requires only around 3s, while the two deterministic heuristics are extremely fast, producing a solution in less than 0.1s.

Our results point out some guidelines. Before the start of the simulation, a trade-off between the quality of the solutions and time consumption has to be made for the selection of the methods. For a simulation of lower complexity (coarse mesh, low number of size groups, fewer time-steps), *Heuristic 2* is the best alternative to provide a "good enough" solution, obtained in less than a second. As the simulations become more complex and computationally expensive, investing more time at the beginning of the simulation can be extremely beneficial and save significant computational time. Even though the ϵ-MOEA is slower than the two deterministic heuristics, it is advantageous to use it for larger simulations, where even minor improvements in the quality of the solution can translate to large savings in computational time during the simulation.

Table 4: Mean runtime (30 runs, reported in seconds) obtaining orderings for brute force approach, the ϵ-MOEA, and the two heuristic approaches. For the ϵ-MOEA, it refers to 50,000 function evaluations for $P_{5,3}$, $P_{15,3}$ and 150,000 for $P_{30,3}$.

	$P_{5,3}$	$P_{15,3}$	$P_{30,3}$
Brute-force	61.12	/	/
ϵ-MOEA	3.11	149.56	253.21
Heuristic 1	<0.1	<0.1	<0.1
Heuristic 2	<0.1	<0.1	<0.1

5 CONCLUSION AND FUTURE WORK

In this paper, we focus on dynamically utilizing the computational resources allocation of CFD simulations used for modeling bubble size distributions. The resource-constrained symmetric function evaluation problem is formulated as a multi-objective problem. A dynamic selection method of the Pareto-front solutions is proposed that utilizes the existing resources. To solve the multi-objective problem the ϵ-MOEA is applied, and compared with brute-force search and two heuristics that serve as baselines. The methods are tested and compared to three different dimensions of the problem. The results showed that the ϵ-MOEA can successfully approximate the Pareto-front, allowing to utilize the resources optimally at each simulation time-step. This prototype can help practitioners that stumbled upon a similar problem to save computational resources.

The immediate future step of this research is to test the framework in real CFD simulations. Moreover, the MOEAs can be modified and/or combined with the heuristics, to further improve the quality of the solutions. Further improvements can also be made to the Dynamic Selection Method by testing its performance in a diverse set of simulations running on different HPCs.

ACKNOWLEDGMENTS

This work was financially supported by the Helmholtz European Partnering Program in the project Crossing borders and scales (CROSSING) and Slovenian Research Agency (research core funding No. P2-0098 and young researcher grant).

REFERENCES

[1] Fabio Banchelli, Marta Garcia-Gasulla, Guillaume Houzeaux, and Filippo Mantovani. 2020. Benchmarking of state-of-the-art HPC Clusters with a Production CFD Code. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. 1–11.

[2] Tobias Benecke. 2020. *Tracing the impact of the initial population in evolutionary algorithms.* Ph. D. Dissertation. Otto-von-Guericke Universität, Magdeburg, Germany.

[3] Jürgen Branke, Kalyanmoy Deb, Henning Dierolf, and Matthias Osswald. 2004. Finding knees in multi-objective optimization. In *International conference on parallel problem solving from nature*. Springer, 722–731.

[4] Stewart Cant. 2002. High-performance computing in computational fluid dynamics: progress and challenges. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 360, 1795 (2002), 1211–1225.

[5] Pei Chann Chang, Shih Hsin Chen, Qingfu Zhang, and Jun Lin Lin. 2008. MOEA/D for flowshop scheduling problems. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 1433–1438.

[6] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. 2000. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, Marc Schoenauer, Kalyanmoy Deb, Günther Rudolph, Xin Yao, Evelyne Lutton, Juan Merelo, and Hans-Paul Schwefel (Eds.). Lecture Notes in Computer Science, Vol. 1917. Springer Berlin / Heidelberg, 849–858. https://doi.org/10.1007/3-540-45356-3_83

[7] Kalyanmoy Deb, Manikanth Mohan, and Shikhar Mishra. 2005. Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions. *Evolutionary computation* 13, 4 (2005), 501–525.

[8] Pedro A. Diaz-Gomez and Dean F. Hougen. 2007. Initial Population for Genetic Algorithms: A Metric Approach. In *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods, GEM 2007*, Hamid R. Arabnia, Jack Y. Yang, and Mary Qu Yang (Eds.). CSREA Press, 43–49.

[9] Susann Hänsch, Dirk Lucas, Eckhard Krepper, and Thomas Höhne. 2012. A multi-field two-fluid concept for transitions between different scales of interfacial structures. *International Journal of Multiphase Flow* 47 (2012), 171–182.

[10] G Houzeaux, RM Badia, R Borrell, D Dosimont, J Ejarque, M Garcia-Gasulla, and V López. 2021. Dynamic resource allocation for efficient parallel CFD simulations. *arXiv preprint arXiv:2112.09560* (2021).

[11] Yacine Kessaci, Nouredine Melab, and El-Ghazali Talbi. 2011. A pareto-based GA for scheduling HPC applications on distributed cloud infrastructures. In *2011 International Conference on High Performance Computing & Simulation*. IEEE, 456–462.

[12] Saku Kukkonen and Jouni Lampinen. 2005. GDE3: The third evolution step of generalized differential evolution. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 1. IEEE, 443–450. https://doi.org/10.1109/CEC.2005.1554717

[13] Ronald Lehnigk, William Bainbridge, Yixiang Liao, Dirk Lucas, Timo Niemi, Juho Peltola, and Fabian Schlegel. 2021. An open-source population balance modeling framework for the simulation of polydisperse multiphase flows. *AIChE Journal* (2021), e17539.

[14] Jing Liu, Xing-Guo Luo, Xing-Ming Zhang, Fan Zhang, and Bai-Nan Li. 2013. Job scheduling model for cloud computing based on multi-objective genetic algorithm. *International Journal of Computer Science Issues (IJCSI)* 10, 1 (2013), 134.

[15] Hongmei Lyu, Fabian Schlegel, Roland Rzehak, and Dirk Lucas. 2020. Improvement of Euler-Euler simulation of two-phase flow by particle-center-averaged method. In *14th International Conference on CFD in 6 Oil & Gas, Metallurgical and Process Industries SINTEF, Trondheim, Norway, October 12–14, 2020*. SINTEF Academic Press.

[16] Máximo Méndez, Mariano Frutos, Fabio Miguel, and Ricardo Aguasca-Colomo. 2020. Topsis decision on approximate pareto fronts by using evolutionary algorithms: Application to an engineering design problem. *Mathematics* 8, 11 (2020), 2072.

[17] Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B Hall, Christopher H Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O Twardziok, Alexander Kanitz, et al. 2021. Sustainable data analysis with Snakemake. *F1000Research* 10 (2021).

[18] Wei Peng, Qingfu Zhang, and Hui Li. 2009. Comparison between MOEA/D and NSGA-II on the multi-objective travelling salesman problem. In *Multi-objective memetic algorithms*. Springer, 309–324.

[19] Gašper Petelin, Ronald Lehnigk, Jeffrey Kelling, Gregor Papa, and Fabian Schlegel. 2021. GPU-based Accelerated Computation of Coalescence and Breakup Frequencies for Polydisperse Bubbly Flows.

[20] Fabian Schlegel, Mazen Draw, Ilya Evdokimov, Susann Hänsch, Harris Khan, Ronald Lehnigk, Jiadong Li, Hongmei Lyu, Richard Meller, Gašper Petelin, and Matej Tekavčič. 2021. HZDR Multiphase Addon for OpenFOAM (Version 2.1.1). Rodare. https://doi.org/10.14278/rodare.1133 http://doi.org/10.14278/rodare.1133.

[21] Jannike Solsvik and Hugo A Jakobsen. 2015. The foundation of the population balance equation: a review. *Journal of Dispersion Science and Technology* 36, 4 (2015), 510–520.

[22] Sergi Vila, Fernando Guirado, Josep L Lerida, and Fernando Cores. 2019. Energy-saving scheduling on IaaS HPC cloud environments based on a multi-objective genetic algorithm. *The Journal of Supercomputing* 75, 3 (2019), 1483–1495.

[23] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation* 11, 6 (2007), 712–731. https://doi.org/10.1109/TEVC.2007.892759

[24] E. Zitzler, M. Laumanns, and L. Thiele. 2001. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems* (19–21 September 2001), K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty (Eds.). International Center for Numerical Methods in Engineering, Athens, Greece, 95–100.

A EMPIRICAL STUDY OF THE MOEAS

In this Appendix, we present the empirical studies of the experiments from five different MOEAs. Table 5 reports the mean objective values obtained using two proposed deterministic heuristics and five MOEAs. We can observe that all of the five MOEAs outperform the proposed heuristics. Figure 8 compares the convergence of the different MOEAs on problems $P_{5,3}$, $P_{15,3}$ and $P_{30,3}$.

Table 5: Mean objective values obtained by both heuristics and all five MOEAs. Values for MOEAs are reported as a mean over 30 runs.

Problem	Algorithm	CPU resources	GPU resources
$P_{5,3}$	Heuristic 1	2706.4	2983.0
	Heuristic 2	1783.2	2195.8
	SPEA2	1707.5	1565.5
	GDE3	1719.0	1563.8
	ϵ-MOEA	1707.7	1568.2
	NSGA-II	1742.0	1541.9
	MOEA/D	1715.3	1562.2
$P_{15,3}$	Heuristic 1	24086.7	53633.3
	Heuristic 2	13420.2	31984.3
	SPEA2	13550.5	18146.6
	GDE3	13693.5	18336.1
	ϵ-MOEA	13421.6	17827.7
	NSGA-II	13514.1	17804.5
	MOEA/D	13734.7	18416.8
$P_{30,3}$	Heuristic 1	125055.1	167591.8
	Heuristic 2	61618.6	98636.1
	SPEA2	59447.7	57395.7
	GDE3	59315.4	57445.8
	ϵ-MOEA	58412.6	56427.1
	NSGA-II	58543.5	56758.9
	MOEA/D	58643.9	57211.5

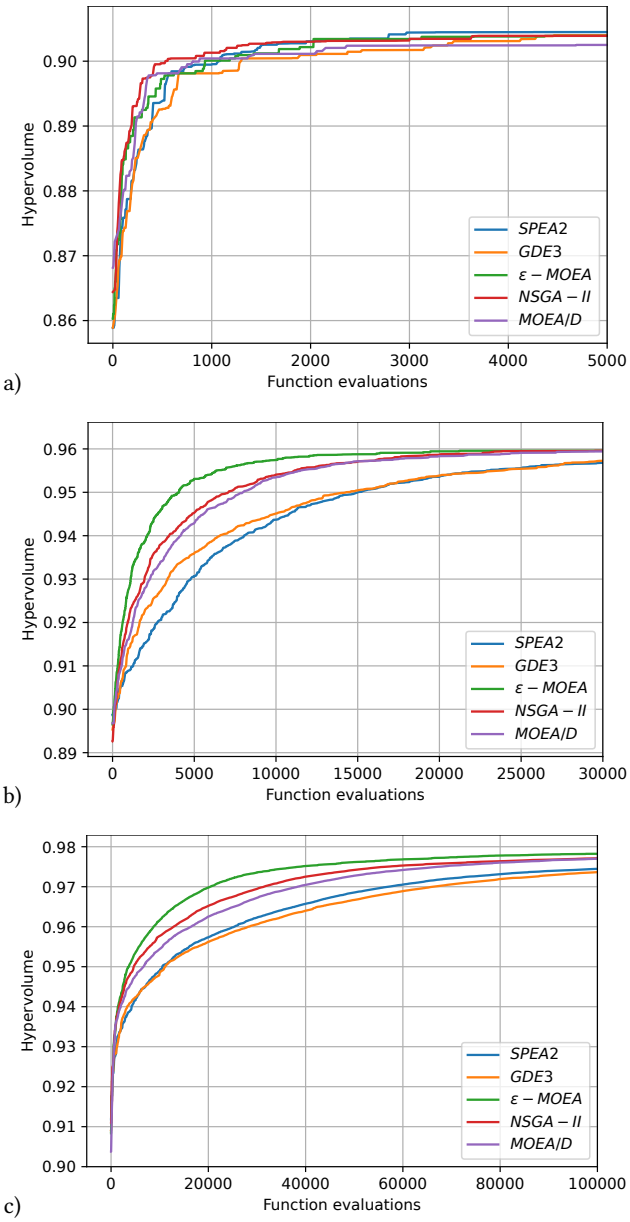

Figure 8: Hypervolume of different MOEAs for problems a) $P_{5,3}$, b) $P_{15,3}$ and c) $P_{30,3}$. Note that to better show the convergence, axis are not the same.