Optimization of Execution Parameters of Moldable Ultrasound Workflows under Incomplete Performance Data

Marta Jaros[1][0000−0002−7775−8106] and Jiri Jaros[1][0000−0002−0087−8804]

Brno University of Technology,
Faculty of Information Technology,
Centre of Excellence IT4Innovations,
Brno, Czech Republic
{martajaros,jarosjir}@fit.vutbr.cz

Abstract. Complex ultrasound workflows calculating the outcome of ultrasound procedures such as neurostimulation, tumour ablation or photoacoustic imaging are composed of many computational tasks requiring high performance computing or cloud facilities to be computed in a sensible time. Most of these tasks are written as moldable parallel programs being able to run across various numbers of compute nodes. The number of compute nodes assigned to particular tasks strongly affects the overall execution and queuing times of the whole workflow (makespan) as well as the total computational cost.
This paper employs a genetic algorithm searching for a good resource distribution over the particular tasks, and a cluster simulator evaluating the makespan and cost of the candidate execution schedules. Since the exact execution time cannot be measured for every possible combination of the task, input data size, and assigned resources, several interpolation techniques are used to predict the task duration for a given amount of compute resources. The best execution schedules are eventually submitted to a real cluster with a PBS scheduler to validate the whole technique. The experimental results confirm the proposed cluster simulator corresponds to a real PBS job scheduler with a sufficient fidelity. The investigation of the interpolation techniques showed that incomplete performance data can successfully be completed by linear and quadratic interpolations keeping the maximum mean error below 10%. Finally, the paper introduces a user defined parameter instructing the genetic algorithm to prefer either the makespan or cost, or find a suitable trade-off.

Keywords: task graph scheduling, workflow, genetic algorithm, moldable tasks, makespan estimation, performance scaling interpolation

1 Introduction

All fields of science and engineering use computers to reach new findings, while the most compute power demanding problems require High Performance Computing (HPC) or Cloud systems to give answers to their questions. The problems

being solved nowadays are often very complex and comprise a lot of various tasks with mutual dependencies describing different aspects of the investigated problem. Their computation can be formally described using scientific workflows [2], also referred to as task graphs [22].

Ultrasound computing workflows aim at various applications of the ultrasound such as neurostimation, tumour ablation, targeted drug delivery, or photoacoustic imaging [23]. The goal of ultrasound treatment workflows is to asses the outcome the treatment and adjust the parameters of the ultrasound traducers to deliver the acoustic energy into desired area while preventing any damage to heathy tissue. The goal of photoacoustic imaging is to reconstruct the tissue structure by running an iterative inverse ultrasound models on signals recorded at the body surface [20]. Since the wavelength of the ultrasound signals are very small compared to the investigated area, e.g. human head or chest, and there are tight deadlines by when the simulation outcome has to be delivered, it is necessary to optimize the workflow execution to reduce both the execution time as well as the cost.

The execution of scientific workflows on HPC systems is performed via communication with the HPC front-end, also referred to as the job scheduler [11]. After the workflow data has been uploaded to the cluster, the workflow tasks are submitted to the computational queues where waiting until the system has enough free resources, and all task dependencies have been resolved (predecessor tasks have been finished).

Modern HPC schedulers implement advanced techniques for efficient task and resource management [12]. However, the queuing time, computation time and related cost depend on the task execution parameters provided at submission. These parameters include the required execution time accompanied by the number of compute nodes, cores and accelerators, the amount of main memory and storage space, and more and more frequently, the frequency and power cap of various hardware components. In most cases, only experienced users are endowed by sufficient knowledge to estimate these parameters appropriately knowing the size of the input data for particular tasks. In other cases, default parameters may be chosen leading to inefficient workflow processing.

Complex compute tasks are usually written as moldable distributed programs being able to exploit various amounts and types of computing resources, i.e., they can run on different numbers of compute nodes. However, the moldability is often limited by many factors, the most important of which being the domain decomposition [4] and parallel efficiency (strong scaling) [1]. The goal of the workflow execution optimization is posed as the assignment of suitable amount of compute resources to individual tasks in order to minimize the overall computation time and cost.

While the field of rigid workflow optimization, where the amount of resources per task is fixed or specified by the user in advance, has been thoroughly studied, and is part of common job schedulers such as PBSPro [11] or Slurm [28], the autonomous optimization and scheduling of moldable workflows has still been an outstanding problem, although firstly opened two decades ago in [8].

During the last decade, many papers have focused on the prediction of rigid workflow execution time and enhancing the HPC resource management. For example, Chirkin et al. [5] introduces a makespan estimation algorithm that may be integrated into job schedulers. Robert et al. [22] gives an overview of task graph scheduling algorithms. The usage of genetic algorithms addressing the task scheduling problems has also been introduced, e.g., a task graph scheduling on homogeneous processors using genetic algorithm and local search strategies [13], and performance improvement of the used genetic algorithm [19]. However, a handful works have taken into the consideration the moldability and strong scaling behavior of particular tasks, their dependencies and the current cluster utilization [7, 3, 27].

In all cases, the estimation of the makespan and optimization of the tasks execution parameters rely on the performance database storing strong and weak scaling. However, it is often not possible to benchmark the execution time for all possible combinations of the task type, task inputs and execution parameters. If a task has already been executed with given inputs and execution parameters, the execution time can be retrieved from the performance database. However, for unseen combinations, some kind of interpolation or machine learning techniques have to be used.

In our previous work [16], Genetic Algorithms (GA) [10] and a simple cluster simulator were used to find optimal execution parameters for various workflows on systems with on-demand and static allocations. This paper follows up with our previous work and its main goals are to (1) prove that GA is able to find execution plans for different workflows when using incomplete performance datasets, (2) prove a trade-off parameter to find different solutions meeting contradictory optimization criteria can be introduced, and finally (3) extend the cluster simulator by adding support for backfilling and considering the initial cluster workload. The resilience of the optimization techniques will be investigated on several scenarios and validated against the real workflow makespan measured on the Barbora supercomputer[1].

2 Automatic Optimization of Workflow Execution Parameters

Selection of suitable execution parameters for workflow tasks plays a crucial role in scheduling process and the maskespan/cost optimization. A naive selection of the execution parameters often leads to various unpleasant situations such as unnecessarily long waiting times and idling nodes if high amounts of compute resources were chosen, or on the other hand, premature task termination and crashes if the amount of compute resources was not sufficient.

Even having enough experience with applications used within the workflow, setting the execution parameters properly to get good performance is a difficult and tedious task. The key to get short makespan is to look at the workflow as a

[1] IT4Innovations, Czech republic, `https://docs.it4i.cz/barbora/introduction/`

whole. There are many dependencies among tasks and selection of best execution parameters for each task independently may lead to a suboptimal solution since there is a limited total amount of resources offered by the HPC facilities.

Although batch schedulers implement several optimization methods and heuristics to maintain high cluster utilization and low queueing times, bad execution parameters spoil their submission schedules, e.g., when tens of tasks enter the queue asking for 24 hour allocations but actually finishing after an hour.

2.1 k-Dispatch Workflow Management System

Molding scientific workflows during the scheduling process goes beyond the capabilities of common batch job schedulers which schedules tasks independently only paying attention to their dependencies and requires the resource requirements to be specified in advance. For the modlable workflow scheduling, a workflow management system sitting in between the end user and the batch job scheduler is required [18, 24]. k-Dispatch [18] is a Workflow Management System (WMS) [6, 26] allowing the end users to submit complex workflows with associated data via a simple web interface and have them automatically executed on remote HPC facilities. Although oriented on the ultrasound community and the popular k-Wave acoustic toolbox [24], its general design allows simple adaptation to other workflows and toolboxes by integrating new task graphs, registering new binaries and adding performance tables.

k-Dispatch consists of three main modules depicted in Fig. 1: Web server, Dispatch database and Dispatch core. The user applications, e.g., a stand-alone medical GUI, Web application, or Matlab interface, communicate with the Web server using the secured HTTPS protocol and REST API. The Dispatch database holds all necessary information about the users, submitted workflows, jobs, computational resources, available binaries and the performance data collected over all executed tasks suitable for the execution time estimation. The Dispatch core is responsible for planning, executing and monitoring submitted workflows. The communication with HPC and cloud facilities is done via SSH and RSYNC protocols. For more information, please refer to [18].

2.2 Workflow Optimization within k-Dispatch

The optimization algorithm providing suitable parameters for particular tasks of the workflow is integrated inside the Dispatch core. It is composed of four modules: Optimizer, Estimator, Evaluator and Collector [16].

The Optimizer is based on a Genetic Algorithm implemented in the PyGAD library [9] and its parameter settings have been thoroughly investigated in [16]. The goal of the Optimizer is to generate high quality candidate solutions, each of which holding a list of execution parameters for all tasks in the workflow. In the simplest case, a candidate solution is a vector where the position of the task is given by a breath first traversal through the workflow task graph and the value determines the number of compute nodes to be used. Although several heuristics has been proposed to optimize the execution parameters [7, 14, 27], they have

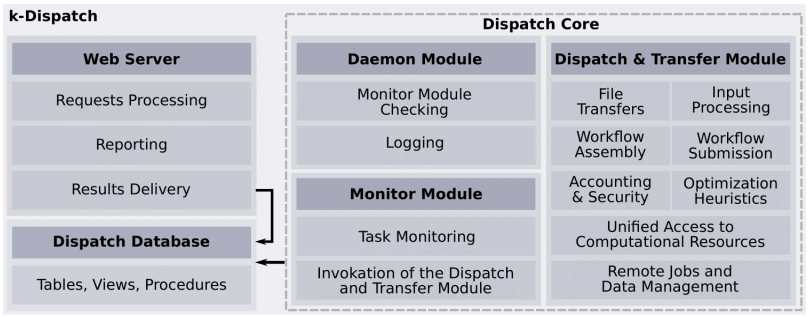

Fig. 1: k-Dispatch's modules and a brief description of the actions each module is responsible for. Arrows show the communication between Dispatch Core, Web Server and Dispatch Database.

strong limitations such as no dependencies among tasks or monotonic strong scaling. The genetic algorithm allows to solve any instance of the optimization problem.

The Estimator is responsible for estimating the execution time for particular tasks based on their input data and the amount of required resources. The Estimator incorporates various interpolation heuristics to reckon up missing values in strong and weak scaling.

The Evaluator uses a simplified simulator of job scheduler called Tetrisator [16], which takes a candidate schedule, simulates its execution on a given cluster and calculates the workflow makespan and cost. Tetrisator is a one-pass simulator of an HPC system with a predefined number of uniform computing nodes. It is inspired by the default strategy of the PBS job scheduler. In this paper, its functionality was extended by the backfilling technique allowing smaller jobs to overtake larger ones if no delays is introduced. The tasks are submitted to the simulator in the order defined in the candidate solution. Workflows may contains multiple dependencies among inner tasks, and the initial cluster workload may be defined, i.e., the cluster is not empty at the workflow submission time.

As soon as a satisfactory solution is found, the workflow is submitted to the real cluster and executed. Upon finishing the execution, the execution times for all tasks are collected by the Collector and stored in the performance database. This data is used to gradually improve the accuracy of the Estimator.

2.3 Estimator Module and Interpolation Techniques

There are many factors that may affect the execution time of a given task. Obviously, the most important ones are the size of the problem stored in the input file and the amount of resources assigned to the task. However, there might be many additional aspects significantly impacting the execution time such as data distribution and load balance, varying time complexity of the algorithms

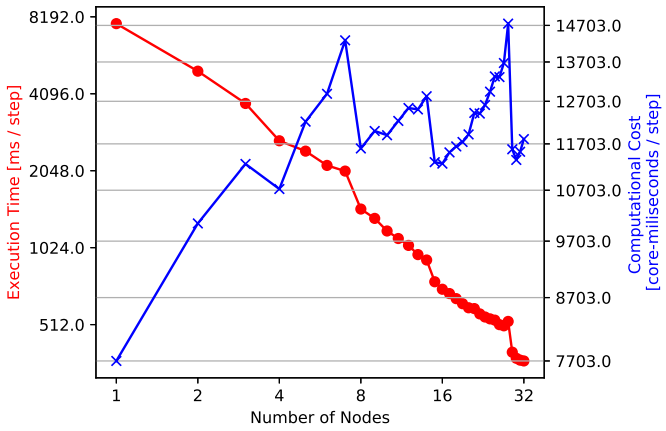

Fig. 2: Red line shows the strong scaling of the k-Wave code measured for a domain size of 1024^3 grid points on the Barbora cluster. Blue line shows the evolution of the computational cost when more nodes are added.

used, additional task parameters and the amount of data being stored during the task execution.

As a practical example, let us talk about the MPI implementation of the k-Wave toolbox [15] simulating (non)-linear propagation of ultrasound wave through a heterogeneous absorbing medium. The scaling of the execution time and cost for one specific problem instance on the Barbora cluster with 36 processor cores per node can be seen in Fig. 2. Here, a domain of 1024^3 grid points is partitioned into 2D slabs and distributed over various numbers of compute nodes (1 to 32). The red curve shows the execution time per one simulation time step (the whole simulation usually executes tens of thousands of time steps). Although this strong scaling curve looks almost ideal, several sudden drops in the execution time can be observed. These drops are the consequences of well balanced workload distribution. For example, if we cut the domain into 512 slices, we can distribute the work over 512 ranks mapped onto 512 cores. Since k-Wave is a memory and network bound application, it is often advantageous to undersubscribe the computing nodes and use higher aggregated memory and network bandwidth. On the Barbora cluster, we can spread 512 ranks over 15 to 28 nodes in a round robin fashion. Since the efficiency of such distribution is decreasing, the scaling curve is flattening toward 28 nodes. However, when 29 nodes are allocated to the task, the domain can be cut into 1024 slices leading to a much better workload distribution and significantly lower execution time. This imperfect workload distribution also renders into the simulation cost since there is a direct proportion between the parallel efficiency and the related cost. The blue curve shows several local minima and maxima in the execution cost which provide very suitable execution parameters or should be avoided, respectively.

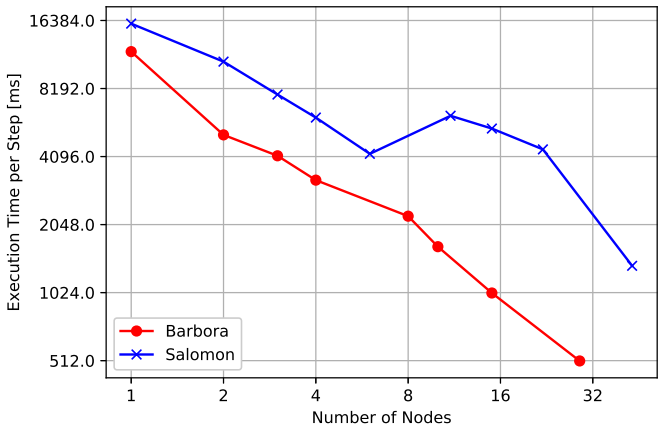

Fig. 3: Strong scaling of the k-Wave code execution time measured for 1024^3 domain size on the Barbora (36 cores/node) and Salomon (24 cores/node) cluster.

Let us note that having a complete performance dataset with all possible input sizes, numbers of nodes, and other tens of simulation parameters is computationally intractable. When having incomplete performance datasets where some points on the scaling curve are missing, the interpolation should rather overestimate the execution time to prevent premature task termination. Even more important question is how the scaling curve changes when a previously unseen domain size is used. In this situation, it is necessary to estimate both the shape and the position of the scaling curve from measured strong and weak scaling. As interpolation functions, linear and quadratic interpolation were used.

Finally, the scaling curves may change significantly among different machines. One such an example can be seen in Fig. 3 where the same problem is solved on Barbora (36 Cascade Lake cores per node) and Salomon (24 Haswell cores per node). Not only is the curve shifted due to a lower node performance, but it has a very different shape in the second half. This may be the effect of a different interconnection network topology, but also current cluster utilization. In this case, it may be very hard to use any interpolation. Thus when a new cluster is connected to k-Dispatch, a few benchmark runs for the most typical simulation settings are performed to get a minimum amount of performance data.

2.4 Evaluator Module Improvement

Since our previous work [16], Tetrisator has been extended by implementing the backfilling technique [21] to simulate the real batch scheduler more accurately.

Tetrisator schedules tasks in the same order as they come to the HPC system (breadth first traversal of the workflow) [16]. The tasks are waiting in an entry queue until executed. A task at the front of the entry queue is ready for execution

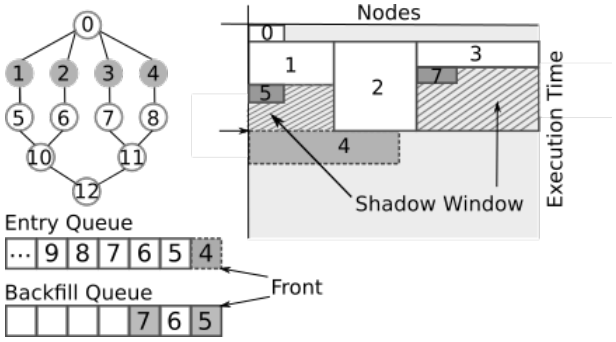

Fig. 4: The state of the entry and backfill queues when the task no 4 is about to be executed. Workflow tasks came to the entry queue in the execution order. Tasks 0-3 have already been executed. When task 4 is to be executed, the scheduler (depicted as schedule on the right) gets short of free nodes, and allows task 5 and 7 to overtake task 4 and fill the *shadow window*.

when all the task dependencies have been fulfilled and there are enough free resources. If this condition is not met, backfilling may find its place. Provided that the execution time of the waiting task will not be postponed and any task dependencies will not be offended, tasks requiring smaller amount of resources may overtake the waiting task. The task to be backfilled is calculated in a so called *shadow window*. The implemented algorithm is depicted in Fig. 4.

Real batch schedulers[2] may implement more sophisticated criteria for backfilling. For example, user and task priorities may be taken into account when deciding which tasks may overtake the waiting ones (fair-share policy). Since we mainly aim at static allocations where users do not compete, this calculation is omitted, i.e., all tasks and users have the same priority and only task dependencies are considered.

The implemented backfilling algorithm considers a queue of jobs that could be possibly backfilled, i.e., the *backfill queue* of length n and width of 1. n stands for a positive number of tasks that have the capability to be backfilled. Width of 1 means that the dependent tasks on the direct candidates to backfill are not considered. In other words, let us have task A actually being calculated, task B waiting for task A and other two tasks C and D. Task D depends on task C. Task C is not dependant on any other task and since not offending the execution time of task B, it can be directly added to the backfill queue. Task D could be also executed and finished within the *shadow window* as task C as well as still not running out of available resources, however, since dependent on task C it is not added to the *backfill queue* (attacking width of 2).

[2] https://docs.it4i.cz/general/job-priority/

3 Experiment Setup

This paper follows the experimental setup presented in [16] to evaluate the developed workflow schedules under incomplete performance database. For the makespan and cost evaluation, the Tetrisator simulator worked with a 54 node cluster. The validation of the final schedules was performed on the Barbora cluster, where a static allocation was created to ensure the same initial conditions for all tests.

3.1 Investigated Workflows

This paper uses two typical biomedical ultrasound workflows applied in the ultrasound neurostimulation and photoacoustic imaging, see Fig. 5. Both workflows consist of two types of tasks. The simulation tasks (ST) executing the k-Wave MPI solver represent heavy parallel jobs running for a few hours. The ST tasks were limited to use between 1 and 32 nodes (36 - 1152 cores). The data processing tasks (PT) perform data pre-processing, post-processing, aggregation, etc. The PT tasks have a linear time complexity and almost perfect scaling. Since their runtime is on the order of minutes, only one or two nodes depending on the amount of memory requested are used.

The first workflow starts with a single PT task generating input files for the ST tasks. Consequently, a few independent trains of ST-PT-ST tasks are executed. Finally, the results from all trains are aggregated using a parallel reduction tree composed of PT tasks. The second workflow starts by running a few ST tasks operating on the same input file, but with different parameters. The results are aggregated into a single output file using a parallel tree reduction. But this time, the result is used by the following wave of ST tasks. In practise, this workflow is repeated in a loop until some error metric calculated by the last PT task is satisfied.

3.2 Used Datasets

Let us here define the datasets used in our experiments along with their short description:

- **Dataset A**. Reference strong scaling of the k-Wave code measured on a domain size of $1024 \times 1024 \times 1024$ grid points using 1-32 nodes.
- **Dataset 1A**. Based on *Dataset A* but having only 16 values including peaks and values in between them.
- **Dataset 2A**. Based on *Dataset A* but having only 8 values excluding peaks.
- **Dataset B**. Reference strong scaling of the k-Wave code measured on a domain size of $810 \times 810 \times 810$ grid points using 1-32 nodes.
- **Dataset 1B**. $810 \times 810 \times 810$ domain interpolated for 1-32 nodes using the quadratic interpolation from the known domain sizes: $512 \times 512 \times 512$, $648 \times 648 \times 648$, $1024 \times 1024 \times 1024$.

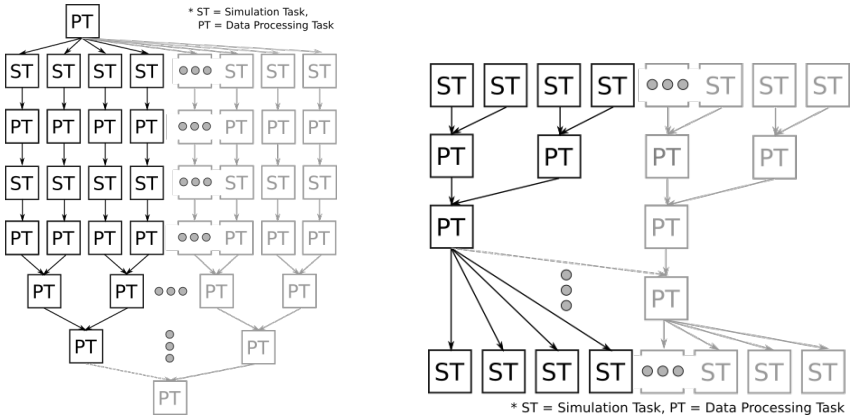

Fig. 5: The structure of investigated workflows. The heavy simulation tasks are interleaved with light data processing tasks. The parts highlighted in black show the minimal workflow structure consisting of 20 and 11 tasks, respectively. The parts displayed in grey show how the workflow structure can grow.

3.3 Tetrisator Validation against Real Cluster

To compare the simulator output with the real execution carried out in a dedicated queue comprising 54 nodes of the Barbora cluster, an artificial schedule based on the first workflow type was created. This workflow contained 20 tasks, (8 heavy STs alternated with 12 light PTs). The execution times of particular tasks were taken from the *Dataset A*. The number of simulation time steps inside the ST tasks were reduced to make the workflow finish in less that 1 hour. To prevent premature termination, a safety cap of 10% calculated from the estimated execution time was added to each task. The real execution time actually covers net computing time as well as overheads such as the computing node initialization. Performed experimental scenario expects no initial workload, i.e., the cluster was empty when the workflow was submitted and executed. The obtained experimental results are then compared against two evolved execution plans employing Tetrisator with backfilling switched on and off.

3.4 Workflow Schedule Quality Measures

The quality of the developed workflow schedules is evaluated by a fitness function the Optimizer calls after the execution trace has been created by Tetrisator. This work investigates two different fitness functions: GODA and GOSA.

GODA (Global Optimization of the workflow on systems with on-Demand Allocations) calculates the makespan over the longest critical path including queueing times. However, the execution cost considers only truly consumed resources. This is a typical cluster operation with users competing for resources. Since having two contradictory criteria, a user-defined scalarization parameter α

is used to balance between the execution time and cost. The algorithm cannot perform a true multi-objective optimization because there is no further feedback from the user that could select the preferred solution from the Pareto front. Contrary, the most suitable solution has to be chosen autonomously and submitted to the cluster as soon as possible (before the cluster background workload changes significantly).

GOSA (Global Optimization of the workflow on systems with Static Allocations) expects the user holds a dedicated part of the cluster and thus has to pay for the whole allocation no matter some nodes may remain idle. Although this is a more expensive solution, it usually reduces the queueing time. Since the makespan and cost are directly proportional, no scalarization coefficient is needed and only the makespan is considered.

3.5 Evaluation of Interpolation Techniques

To estimate missing execution time for a particular task, domain size, and number of nodes, two different interpolation techniques from the Python's *scipy* package [25] were used. After a thorough investigation in [17] and new experiments performed in the paper, a linear and quadratic `interp1d` interpolations were chosen. Very similar results to the quadratic interpolation were also obtained by cubic spline `CubicSpline` with the `bc_type` parameter set to `natural`. Unfortunately, the use of the default value of `bc_type` caused high oscillations and strong underestimations of the execution time. Therefore, we decided to use a quadratic interpolation instead.

Three different experiments with the interpolation functions were conducted. The goals of particular experiments were

- to estimate missing points on the strong scaling curve for a domain size of 1024^3 grid points defined by the points with ideal scaling ($N\%(P*36) \approx 0$), where N is the domain size and P is the number of nodes, see Fig. 7.
- to estimate missing points on the strong scaling curve for a domain size of 1024^3 grid points when having also points in the middle of the intervals between two points with ideal scaling, see Fig. 7.
- to reconstruct a completely unknown scaling curve for an unseen domain size from the data stored in the performance database. In this example, scaling curves for 512^3, 648^3 and 1024^3 were used to estimate the one for 810^3 grid points, see Fig. 8. The domain sizes chosen progressively double the total number of grid points.

As the measure of the interpolation quality, a mean relative error was used, see Eq. (1).

$$meanError = \frac{1}{N}\sum_{i=0}^{N}(\frac{|a_i - b_i|}{a_i}) \tag{1}$$

where a denotes the measured execution time, b the interpolated execution time, and N is the total number of the compute nodes (32).

In all cases, we can tolerate a small overestimation but shall avoid underestimation which leads to premature job termination and necessary resubmission with prolonged execution time.

4 Experimental Results

This section presents and discusses (1) the similarity of the workflow execution schedule to the one executed on a real HPC cluster, and (2) the error reached by the interpolation techniques.

4.1 Simulated Execution Plans Reliability

The following figures point out the differences between simulated execution plans created by Tetrisator and the real executions performed in the dedicated queue on Barbora. Figure 6 shows the scenario where no initial workload is expected and all 54 nodes are fully available at submission time. As expected, the simulated makespan by Tetrisator with backfilling switched off is a bit pessimistic causing the overestimation by 15%. On the contrary, it can be seen that the simulated makespan by Tetrisator using backfilling is underestimated by 3%. This underestimation is, however, caused by cumulative error produced by slight delays of individual task execution times on the cluster.

Our observations suggest that the real PBS cluster scheduler works in the same manner as Tetrisator. This means the tasks within the workflow are submitted to the real cluster in the same order as they are processed by the Tetrisator, and their submission time is more or less the same. Thus, the tasks are also executed one by one in the same manner as arriving to the cluster. The changes in the order happen when a task has to wait for free resources.

4.2 Interpolation Functions Accuracy

Figure 7 shows the measured and interpolated strong scaling curves on a domain composed of 1024^3 grid points. Inspecting the scaling curve created by a linear interpolation, a very close match can be seen. When interpolating using values where the scaling is close to the optimal, the mean interpolation error reaches 4%. After adding the values from the middle of particular intervals, the error drops below 0.8%. Unfortunately, the interpolated values for sparser training data are mostly underestimated, which can be corrected by a small bias or picking the points with the worst instead of best workload distribution.

When repeating the same experiment with a cubic spline and a quadratic interpolation, the mean error gets higher up to the level of 12% and 7%, respectively, depending on the number of known values. The high error is caused by several oscillations, and more specifically, by the extrapolation error where the execution time is extremely underestimated.

The 4% error of the linear interpolation reaches the level of uncertainty of real execution time measurement on clusters due to unstable node, network and I/O

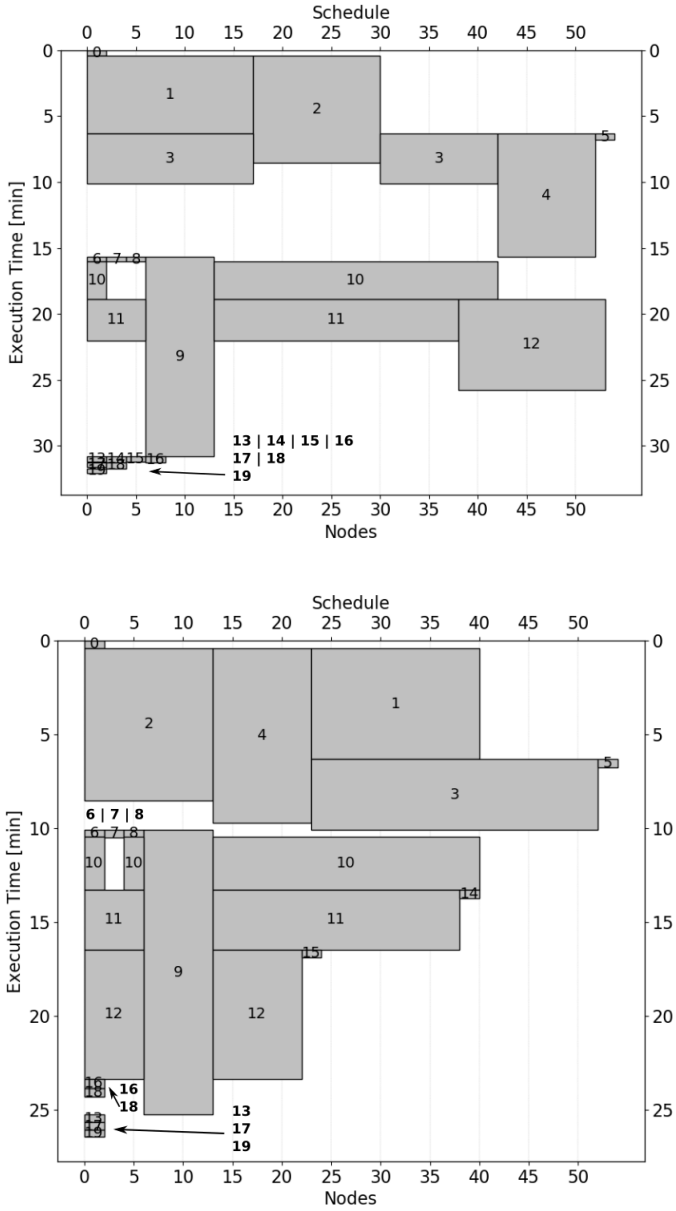

Fig. 6: Two simulated execution schedules. The top one with backfilling switched-off and the makespan reaches 32.1 minutes, and the bottom one implementing backfilling and finishing earlier in 26.4 minutes.

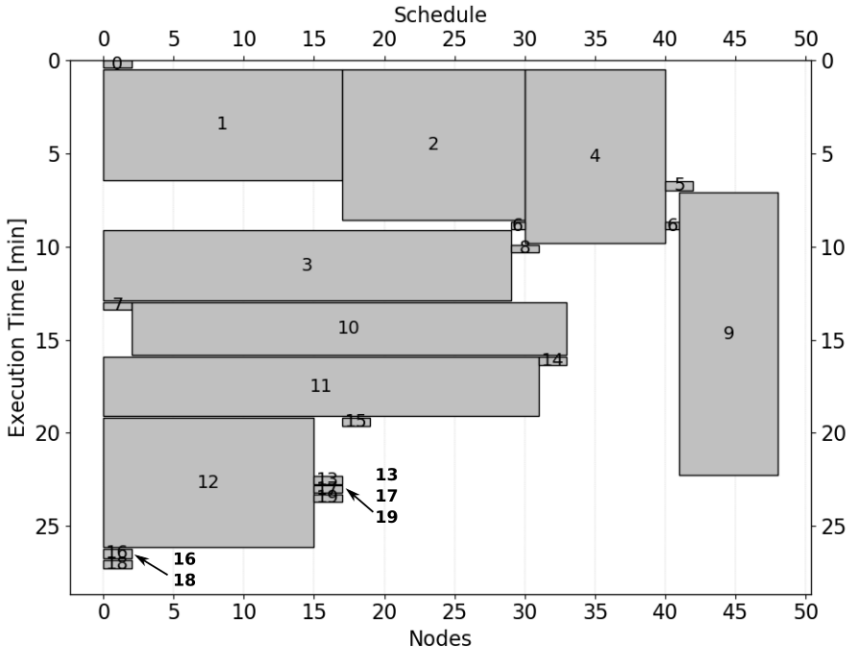

Fig. 6: Real execution of the workflow on Barbora finishing in 27.3 minutes.

performance. The suitability of the linear interpolation can be also attributed to a very good scaling of the ST tasks without any significant anomalies. Since parallel codes have to show good scaling to be deployed in production runs, linear interpolation is expected to work well for most such codes.

The second experiment attempts to estimate the strong scaling for an unknown domain size, see Fig. 8. The figure reveals that the interpolation method rather overestimate the scaling curve. When repeating this experiment with a linear and a natural cubic spline interpolations, we got the mean error of 25.4% and 13.5%, respectively, while the quadratic interpolation and the cubic spline with `bc_type` parameter set to default produced better estimates reaching the mean error at a level of 10.5%. The explanation is quite simple. While the strong scaling of the ST tasks on a given domain size is almost linear, the algorithm has an asymptotic time complexity of $O(n \log n)$. Moreover, the ST tasks heavily employ fast Fourier transform which is very sensitive to the domain size and its prime factors. The quadratic interpolation thus better capture the nature of ST tasks.

The conclusion is to use a linear interpolation to estimate values on known scaling curves while using a quadratic interpolation when the domain size has not been seen before. It is important to say that the k-Wave code is highly tuned and scales very well. Employing a code the scaling of which is more "wild" with many

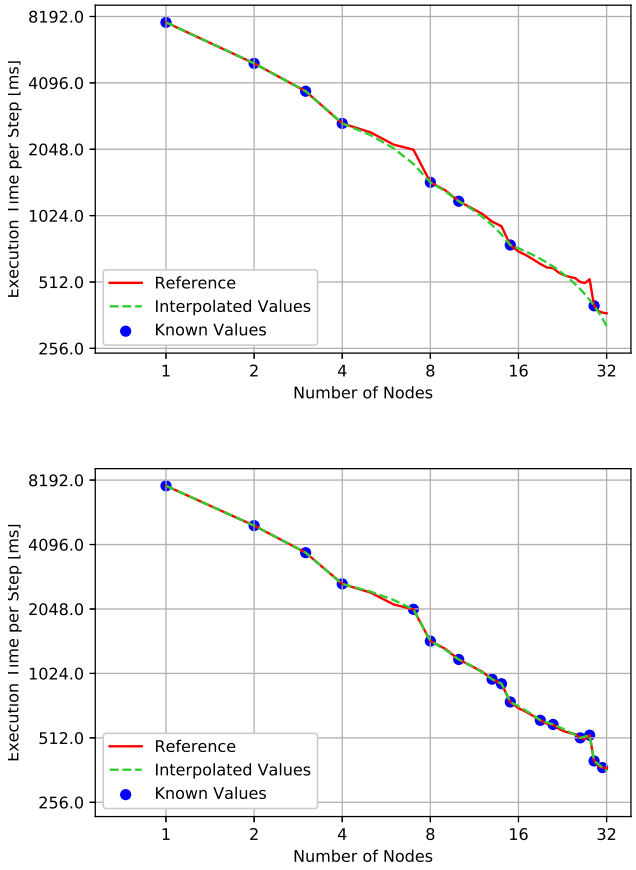

Fig. 7: Reference and interpolated strong scaling of ST tasks for a domain size of 1024^3 grid points with a linear interpolation calculated from 8 and 16 known values, respectively. In the top figure, values in unexpected peaks were selected intentionally to see how much the value would be underestimated.

peaks or a dramatic slowdowns may become a challenge. When using different parallel codes, it may be beneficial to use a different interpolation for unknown domain sizes corresponding to the asymptotic time complexity. Moreover, if the scaling is relatively stable, it may be possible to construct a scaling equation and use a fitting methods to set its coefficients using known performance data. Alternatively, we may try to interpolate the known points using a various polynomial interpolations and based on the error make a decision about a selection of the interpolation method.

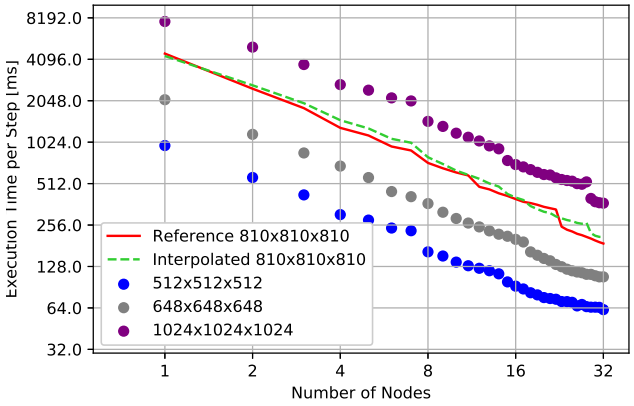

Fig. 8: Reference and interpolated strong scaling of ST tasks for an unknown domain size of 810^3 grid points with a quadratic interpolation.

4.3 Impact of Interpolation on Schedule Makespan and Cost

This section investigates the quality and accuracy of the developed schedules when using the performance database containing all data, only a subset, or no data for particular domain size.

Figure 9 shows the makespan and cost of the best workflow schedules developed for the GODA situation on a known domain size of 1024^3 grid points, with all, 8 and 16 performance values. These experiments also use different values of the α scalarization coefficient (only three values of α are used in figure for better visibility). The schedules were collected over twenty independent runs of the genetic algorithm. The Pareto fronts (lines in the plot) for the same values of α are close to each other confirming that by employing interpolation methods on incomplete datasets we are able to achieve very similar results. When using *Dataset 2A* containing only 8 performance values, the solutions found may be deflected from that ones evolved using dense dataset. This actually does not mean that found solutions are bad, they just overlap the area where solutions for different value of α would be expected. Next, it can be seen that solutions for different α form isolated clusters. This implies we can affect the execution plan to prioritize different criteria. At this point, it is important to note that the execution plan may be adjusted in makespan by a factor of 10.0 while in computational cost by a factor of 1.7. The factors vary and the cost factor is such small due to the highly optimised code used. This is a very promising result showing that when the interpolation is reasonably accurate, the impact on the best solution developed by the Optimizer is rather small.

Table 1 summarizes conducted experiments of GOSA expressing the quality of the execution schedules as makespan. The table may be divided into two parts. The left one is for the domain size of 1024^3 where missing strong scaling

Table 1: The results show GOSA applied on the domain of 1024^3 on the left and 810^3 on the right. Experiments were performed using (1) the full performance dataset without interpolation, (2) the partial performance dataset of 8 and 16 known values, respectively, and completed using linear interpolation, and (3) the full performance dataset created using quadratic interpolation. The table depicts average (Avg), minimum (Min) and maximum (Max) obtained values of makespan in minutes. The percentage difference between experiments with partial and full performance datasets is also depicted.

1024 x 1024 x 1024		40 Tasks Makespan [min]	40 Tasks Diff. [%]	80 Tasks Makespan [min]	80 Tasks Diff. [%]	810 x 810 x 810		40 Tasks Makespan [min]	40 Tasks Diff. [%]	80 Tasks Makespan [min]	80 Tasks Diff. [%]
GOSA	**Avg**	**29.70**	-	**58.31**	-	**GOSA**	**Avg**	**14.82**	-	**30.05**	-
with no	**Min**	27.75	-	55.74	-	**with no**	**Min**	14.07	-	28.32	-
interp.	**Max**	35.10	-	61.07	-	**interp.**	**Max**	16.88	-	31.76	-
GOSA with	**Avg**	**29.19**	**1.72**	**59.23**	**1.57**	**GOSA**	**Avg**	**17.08**	**15.25**	**33.11**	**10.18**
linear interp.	**Min**	27.29	1.65	55.27	0.84	**with quadratic**	**Min**	15.44	9.70	31.27	10.41
(Dataset A1)	**Max**	33.25	5.27	65.47	7.21	**interpolation**	**Max**	18.85	11.64	36.67	15.44
GOSA with	**Avg**	**26.74**	**9.98**	**51.06**	**12.44**						
linear interp.	**Min**	24.87	10.36	49.05	12.00						
(Dataset A2)	**Max**	30.33	13.58	56.46	7.55						

values were completed by a linear interpolation. The right one is for the domain size of 810^3 which was fully interpolated using a quadratic interpolation. The difference between the achieved makespan for the full performance dataset and interpolated datasets is given by an interpolation error (investigated in Sec. 4.2) and performance fluctuations of cluster's nodes. The experiments were provided with both backfilling switch on and off. It turned out backfilling did not impact the results significantly, causing differences between 0.09% and 4%. This suggests the genetic algorithm finds such good workflow schedules that minimize the amount of unused resources so that the backfilling has only a limited space for schedule improvements. Next, a workflow structure also influences how good the workflow could be mapped.

5 Conclusions

The paper has investigated the optimization of moldable ultrasound workflow executions under incomplete performance database where the execution times for some combination of tasks, input data and amount of resources are not known and have to be estimated. Consequently, the paper has proven the workflow execution on a cluster can be simulated and this simulator can be integrated in the k-Dispatch's optimization module. Although being a one-pass PBS-based simulator, the estimations provided are sensible. The simulator gives accurate estimations especially for workflows executed on dedicated resources where other workload is known. The cross validation of an artificial and the real schedules created by the PBS job scheduler on Barbora show a good general match.

The experimental results indicate that linear interpolation works well in situations the input data has been seen before and the task has already been executed using a few execution parameters configurations. In such cases, the

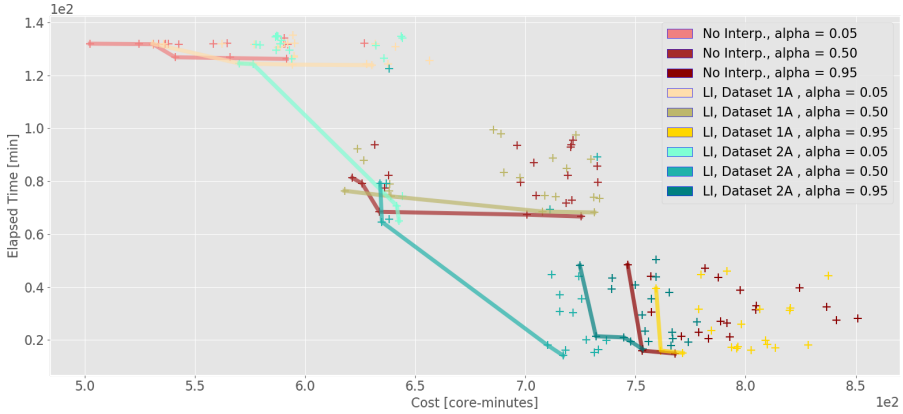

Fig. 9: Pareto front together with dominated solutions showing the evolved schedules for workflows of 11 tasks not requiring interpolation, and two experiments both using linear interpolation (LI) but differing in the content of the performance dataset.

missing performance data can be calculated with a very small error below 4%. From our experience, linear interpolations appear to be generally applicable on parallel codes with good strong scaling. On the other hand, if the input data has not been seen before, the execution time has to be estimated from similar inputs by interpolating between known strong/weak scaling curves. In this case, a quadratic interpolation worked sufficiently well for our codes, however, the error may reach 10%. This can be attributed to used codes having $O(N \log N)$ time complexity. For codes with different time complexity, higher polynomial interpolations may produce better results.

The paper also confirms that it is possible to find different schedules that prioritize various criteria using the trade-off parameter α. The proposed optimization algorithm constructs the Pareto front offering different suitable schedules. Users, however, (1) are not aware of what tasks are executed within the workflow, (2) may not know what solution to choose, and finally (3) the Pareto fronts are calculated just before the workflow execution and this information is not available at submission time to k-Dispatch. This is the reason why a multi-criteria optimization is transformed to an easier form where users can express their preferences between two criteria (makespan vs. computational cost) using, e.g., a slider bar just before workflow submission to k-Dispatch.

The developed schedules tend to overestimate the execution time, which is partially caused by imperfect interpolation, and a reserve of 10% added to the workflow to avoid premature termination. Nevertheless, the error between developed and real schedules fits within a 15% margin, which is considered to be acceptable for most users.

5.1 Future Work

There are two directions we would like to follow in our future work. First, we would like to include the information about the actual cluster utilization into the cluster simulator. This will allow us to better simulate workflow execution in on-demand allocations where the user competes with others. It may have an impact on the shape of the developed schedules because tasks asking for more resources sit longer in the queue. Using smaller amounts of resources thus may improve the workflow makespan. Second, we would like to examine more advanced machine learning techniques to improve the interpolation accuracy once the performance database includes tens of thousands of records.

Acknowledgments. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140). This work was supported by Brno University of Technology under project numbers IGA FIT/FSI-J-22-7980 Acceleration of Selected Evolutionary Communication Techniques for Solving Combinatoric Tasks and FIT-S-20-6309 Design, Optimization and Evaluation of Application Specific Computer Systems.

References

1. G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 1820 1967 spring joint computer conference*, 23(4):483–485, 1967.
2. S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. In *Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10. IEEE, 2008.
3. R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mounie, and D. Trystram. Scheduling Independent Moldable Tasks on Multi-Cores with GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 28(9):2689–2702, sep 2017.
4. T. F. Chan and T. P. Mathew. Domain decomposition algorithms. *Acta Numerica*, 3:61–143, jan 1994.
5. A. M. Chirkin, A. S. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov. Execution time estimation for workflow scheduling. *Future Generation Computer Systems*, 75, 2017.
6. E. Deelman, K. Vahi, G. Juve, et al. Pegasus: a workflow management system for science automation. *Future Generation Computer Systems*, 2014.
7. P.-F. Dutot, M. A. S. Netto, A. Goldman, and F. Kon. Scheduling Moldable BSP Tasks. In *Lecture Notes in Computer Science*, pages 157–172. 2005.
8. D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Lecture Notes in Computer Science*, pages 1–26. 1996.
9. A. F. Gad. Geneticalgorithmpython: Building genetic algorithm in python, 2021.
10. D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley Longman, Boston, MA, 1989.
11. R. L. Henderson. Job scheduling under the Portable Batch System. In *Lecture Notes in Computer Science*, pages 279–294. 1995.

12. M. Hovestadt, O. Kao, A. Keller, and A. Streit. Scheduling in HPC resource management systems: Queuing vs. planning. *Lecture Notes in Computer Science*, 2862(June):1–20, 2003.
13. H. Izadkhah. Learning based genetic algorithm for task graph scheduling. *Applied Computational Intelligence and Soft Computing*, 2019.
14. K. Jansen and F. Land. Scheduling Monotone Moldable Jobs in Linear Time. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 172–181. IEEE, may 2018.
15. J. Jaros, a. P. Rendell, and B. E. Treeby. Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. *International Journal of High Performance Computing Applications*, 30(2):137–155, 2016.
16. M. Jaros and J. Jaros. Performance-cost optimization of moldable scientific workflows. In *Job Scheduling Strategies for Parallel Processing. JSSPP 2021*. Springer Nature Switzerland AG, 2021.
17. M. Jaros, T. Sasak, E. B. Treeby, and J. Jaros. Estimation of execution parameters for k-wave simulations. In *High Performance Computing in Science and Engineering*, pages 116–134. Springer, 2020.
18. M. Jaros, E. B. Treeby, J. Jaros, and P. Georgiou. k-dispatch: A workflow management system for the automated execution of biomedical ultrasound simulations on remote computing resources. In *Platform for Advanced Scientific Computing Conference*, pages 1–10. ACM, 2020.
19. F. A. Omara and M. M. Arafa. Genetic algorithms for task scheduling problem. *Journal of Parallel and Distributed Computing*, 70(1):13–22, 2010.
20. J. Poudel, Y. Lou, and M. A. Anastasio. A survey of computational frameworks for solving the acoustic inverse problem in three-dimensional photoacoustic computed tomography. *Physics in Medicine and Biology*, may 2019.
21. H. Rajaei and M. Dadfar. Comparison Of Backfilling Algorithms For Job Scheduling In Distributed Memory Parallel System. In *2006 Annual Conference and Exposition Proceedings*, pages 11.339.1–11.339.12. ASEE Conferences, 2007.
22. Y. Robert. Task Graph Scheduling. *Encyclopedia of Parallel Computing*, pages 2013–2025, 2011.
23. T. L. Szabo. Diagnostic Ultrasound Imaging: Inside Out, 2014.
24. B. Treeby and B. Cox. k-wave: Matlab toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of biomedical optics*, 15:021314, 03 2010.
25. P. Virtanen, R. Gommers, and T. E. a. o. Oliphant. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
26. K. Wolstencroft, R. Haines, D. Fellows, et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 5 2013.
27. D. Ye, D. Z. Chen, and G. Zhang. Online scheduling of moldable parallel tasks. *Journal of Scheduling*, 21(6):647–654, 2018.
28. A. B. Yoo, M. A. Jette, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In *Lecture Notes in Computer Science*, pages 44–60. 2003.