Estimation of Execution Parameters for k-Wave Simulations

Marta Jaros[1][0000−0002−7775−8106], Tomas Sasak[1], Bradley E. Treeby[2][0000−0001−7782−011X], and Jiri Jaros[1][0000−0002−0087−8804]

[1] Brno University of Technology, Faculty of Information Technology,
Centre of Excellence IT4Innovations,
Bozetechova 2, 612 66 Brno, Czech Republic
{martajaros,jarosjir}@fit.vutbr.cz,
xsasak01@stud.fit.vutbr.cz

[2] University College London, Medical Physics and Biomedical Engineering,
Biomedical Ultrasound Group,
Malet Place Eng Bldg, WC1E 6BT London, United Kingdom
b.treeby@ucl.ac.uk

Abstract Estimation of execution parameters takes centre stage in automatic offloading of complex biomedical workflows to cloud and high performance facilities. Since ordinary users have no or very limited knowledge of the performance characteristics of particular tasks in the workflow, the scheduling system has to have the capabilities to select appropriate amount of compute resources, e.g., compute nodes, GPUs, or processor cores and estimate the execution time and cost.
The presented approach considers a fixed set of executables that can be used to create custom workflows, and collects performance data of successfully computed tasks. Since the workflows may differ in the structure and size of the input data, the execution parameters can only be obtained by searching the performance database and interpolating between similar tasks. This paper shows it is possible to predict the execution time and cost with a high confidence. If the task parameters are found in the performance database, the mean interpolation error stays below 2.29%. If only similar tasks are found, the mean interpolation error may grow up to 15%. Nevertheless, this is still an acceptable error since the cluster performance may vary on order of percent as well.

Keywords: Workflow management system · performance data collection · interpolation · job scheduling · HPC as a service.

1 Introduction

Computation of complex scientific applications may no longer be satisfied by personal computers and small servers manually operated by highly experienced users. First, the extent of data being processed and the computational requirements highly exceed the capacity of such machines. Increasing number of applications is thus moving to the cluster or cloud environments. Second, scientific applications often feature a very complex processing workflows consisting of many

particular tasks employing different computer codes, and complex data dependencies. Third, scheduling, execution and monitoring of such workflows require automated tools to remove the burden from the experienced users, enable ordinary users to routinely execute their applications, and increase the throughput of the computing facilities.

To face these challenges, the scientific and software development communities have adopted the workflow paradigm to describe the processing flow. The most common formalism used is the weighted directed acyclic graph (DAG) defining computational tasks by the nodes, and the dependencies and data movements by the edges. The weights in the nodes describe the computational requirements while the weights on the edges denote the amount of data being transferred between tasks [15].

In order to automate workflow execution, several workflow management systems (WMSs) have been developed and used within the scientific community. The most popular tools such as Pegasus [2,3], Globus [4] or Kepler [12] now offer automated execution of scientific workflows on remote computational resources in a more or less general way. However, these tools focus on expert users who know the behaviour of the computational codes used within the workflow, and are able to estimate the amount of computational resources needed by each task. The scheduling and mapping of the workflow on the computational resources are usually left to the cluster batch processing systems such as PBS[3] or Slurm[4].

These task schedulers provide their best effort to execute the tasks in the earliest possible time depending on the cluster workload and user/task priorities. However, what they cannot deal with is the execution parameters settings. If the user overestimates the amount of the computing resources, the tasks may be waiting in the queue for much longer time while making only little benefit from increased amount of resources, e.g., processor cores. On the other hand, underestimating these requirements may lead to the premature task termination due to exhausting the execution time.

This paper focuses on the heuristic-based selection of the execution parameters for a list of predefined computing codes used in the biomedical workflows supported by the k-Wave toolbox [18]. Since all binaries are fixed and known in advance, their performance characteristics such as strong and weak scaling can be automatically collected and used for prediction. Limiting the users in uploading their binaries also enables fine-grain performance tuning of the underlying codes for target machines and simplifies the workflows composition by the use of high-level processing blocks.

The next section describes the k-Plan system supporting the design of ultrasound workflows via a graphical user interface, and workflow offloading, scheduling, execution and monitoring using the k-Dispatch module. Section 3 describes a single pass optimization of the workflow execution parameters and related interpolation heuristics. Section 4 investigates the quality of interpolation for known and unknown tasks, and Section 5 concludes the paper.

[3] https://www.altair.com/pbs-works/
[4] https://slurm.schedmd.com/

2 Automatic Offloading of k-Wave Workflows

The k-Wave toolbox [18] is an open source Matlab toolbox designed for the time-domain simulation of acoustic waves propagating in tissues. The toolbox has a wide range of functionality, but at its heart is an advanced numerical model that can account for both linear and nonlinear wave propagation, an arbitrary distribution of heterogeneous material parameters, power law acoustic absorption and its thermal effects on the tissue. During recent years, k-Wave has attracted a lot of attention amongst biomedical physicists, ultrasonographers, neurologists and oncologists. Many k-Wave-based applications have been reported in photoacoustic breast screening [13], transcranial brain imaging [14], and high intensity focused ultrasound treatment planning for kidney [1,16], liver [7] or prostate tumour ablations [17].

However, all these applications require very intensive computations. During the last decade, the simulation core has been rewritten in C++ and parallelized by various technologies, such as OpenMP for shared memory systems [19], CUDA for GPU accelerated systems [10], and MPI for large distributed clusters [8]. These implementations now cover a wide range of ultrasound simulations in domains of various sizes reaching the limits of the top supercomputers.

To support clinicians in executing ultrasound workflows, a complex system called k-Plan [9], consisting of tree modules, is being developed, see Fig. 1:

1. TPM - Treatment Planning Module implements user front-end with the graphical user interface to compose the processing workflow. Advanced users may also use a Matlab interface or third-party applications.
2. DSM (k-Dispatch) - Dispatch Server Module is responsible for the workflow offloading to remote computing facilities. It also schedules particular tasks, estimates computing requirements, and monitors the workflow progress.
3. SEM - Simulation Execution Module covers the deployed binaries necessary to run particular tasks. Due to strict medical restrictions, all binaries have to be certified, thoroughly tested and properly deployed.

Although designed for the k-Wave toolbox, k-Dispatch remains as general as possible to support other applications and workflow types. User applications such as TPM communicate with k-Dispatch through the Web server, see Fig. 2. The Dispatch database maintains users and groups, their resource allocations, history of calculated and submitted workflows, available computing facilities, executable binaries with their performance characteristics, etc. Besides decoding the workflows, data transfers, monitoring and communication with remote computing facilities, the k-Dispatch core performs the optimization of the workflow execution parameters.

Users can create new ultrasound procedures by altering predefined workflow templates and packing them with the patient's data. Once delivered to k-Dispatch, the execution workflow is constructed from the provided input file. Next, the list of available computing resources is scanned to find a suitable one, e.g., the one with the lowest actual workload. Consequently, appropriate binaries for particular tasks are filled in to the workflow template according to the tasks

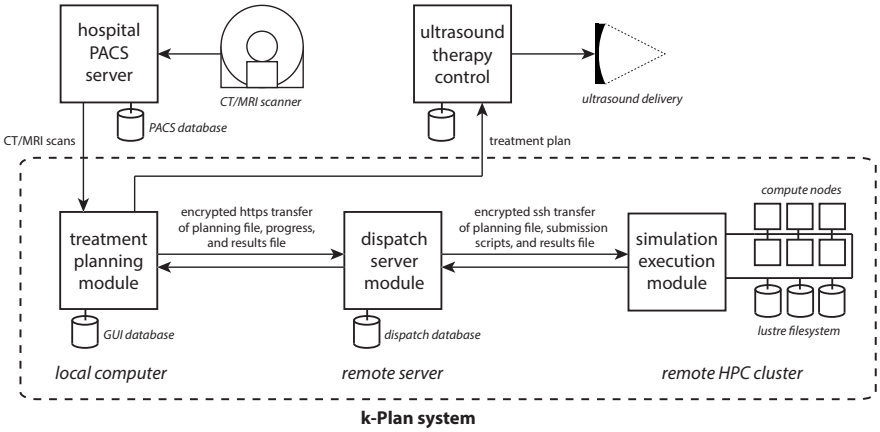


Figure 1: Architecture of the k-Plan system. The dispatch server module (k-Dispatch) arranges for the workload scheduling, execution, monitoring and data transfers between client applications and computing facilities.

input data size and available hardware. Since k-Dispatch knows the performance scaling of the given binaries, it can optimize the amount of computational resources (i.e., number of nodes) assigned to particular tasks and minimize several objectives such as cost, execution time and queuing time, see Algorithm 1.

After the tasks have been submitted to the computational queues, k-Dispatch keeps monitoring them, detects anomalies such as frozen/crashed jobs, and restarts them if necessary. After the workflow computation has been completed, the results are downloaded from the remote computing facility back to the k-Dispatch and the user is notified that the results are available for download.

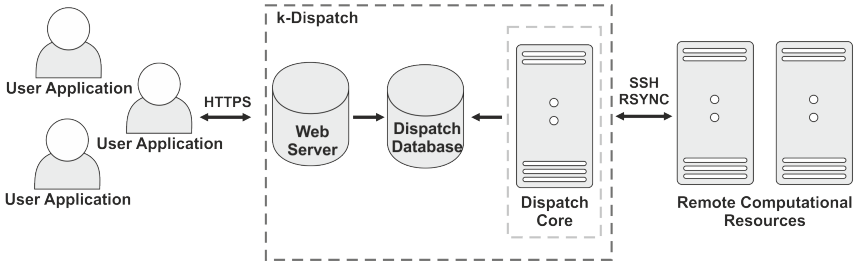


Figure 2: k-Dispatch stands between user applications and remote computational resources. The communication with user applications is based on standard web services while the SSH protocol is used to communicate with remote computational resources. The dispatch core is responsible for the workflow submission, monitoring and other service mechanisms.

Algorithm 1: Adaptive execution planning algorithm

Presumptions :

1 Let $G = (V, E)$ be a workflow where V is a set of tasks and $E \subseteq V \times V$ is a set of task dependencies.
2 Let C be a set of active resource allocations with enough resources to satisfy the workflow G. It holds $C \subseteq A$, where A is a set of all allocations the user has got access to.
3 All executable binaries for supported task types available in a given allocation $a \in A$ are defined as $D \in (B_1, B_2, \ldots, B_N)$, where N is the number of task types within the workflow G, and $B_i = \{b_1, b_2, \ldots, b_M\}$ is the set of available binaries for a given task type. B_i may be an empty set.
4 Let $p : G \times C \times D \to \mathbb{R}^+$ be a price function returning the aggregated computational cost of the workflow G.
5 Let $t : G \times C \times D \to \mathbb{R}^+$ be a function returning the aggregated execution time of the workflow G. This value is calculated as a critical path through the workflow considering both the net execution time e and the queuing time q.
6 Let workflow evaluation f serving as quality metric be defined as $f = \alpha \cdot p + (1 - \alpha) \cdot t$, where α is a selectable ratio prioritizing the minimal computational cost or the execution time.

Algorithm :

1 Create a workflow $G = (V, E)$ from the workflow template and input data.
2 Select a set of candidate allocations $C = \{c \in A^+ \mid c.status == active \wedge c.hours\_left > 0.0\}$.
3 Set appropriate execution parameters for all tasks and evaluate the workflow G for all combinations of candidate allocations C and binary executables D.
4 Return the best parameters for a given workflow G as $\text{argmin}_{(c \in C, d \in D)} f(G)$.

3 Optimization of Workflow Execution Parameters

A typical course an ordinary user takes when executing a complex workflow is to use default execution parameters for each task, often consisting of one computational node and 24 hours of wall time. If a task fails due to insufficient memory or time, another node or more time is allocated and the workflow restarted. Nevertheless, experienced users usually run a few benchmarks with various input sizes and number of nodes to create a strong scaling plot and predict the extent of computational resources for each task, which is the idea k-Dispatch has adopted.

In [9], three levels of workflow optimization were introduced. The naive one using the default execution parameters was implemented to compare k-Dispatch with other WMSs which use firmly set values directly provided by the users. This paper deals with a single-pass, task level optimization, processing each task independently. As we will show later, this is a viable solution with a linear time complexity providing sufficient results when execution cost and time is only considered. However, optimizing also for the queuing time requires a multi-pass, global optimization which may lead to an exponential time complexity, and needs a cluster simulator loaded with actual snapshots of cluster workload.

3.1 Single-pass Optimization

The goal of the single-pass optimization is to independently find such execution parameters for each task i that minimize the workflow evaluation given by

$$f = \sum_{i=1}^{N} (\alpha * p_i + (1 - \alpha) * e_i) \tag{1}$$

where α is a weight preferring either execution cost or time, p is the execution cost and e is the net execution time. The queuing time is omitted here. Currently, the execution parameters to be optimized only cover the number of allocated nodes/cores and the execution time. Nevertheless, it is straightforward to extend the optimization to select the most suitable code, computational queue, node type (accelerated/fat/slim), etc.

Figure 3 illustrates the optimization of the task execution parameters as a black box with a task type and task input file provided by the workflow as the inputs. The task input file is parsed to extract information necessary to estimate the computational requirements. This information typically includes the size of the simulation domain, the simulation timespan, type of the medium, transducer definition, etc. Next, the collected performance data is searched to find similar records. Having a filtered out performance dataset, the plot of strong scaling can be constructed and several interpolation techniques can be used to estimate the task duration and cost for suitable amounts of resources. Once the best execution parameters are selected, the machine specific job scripts are generated and submitted to the computing queue. After the task has been properly finished, the performance data is used to update the performance database.

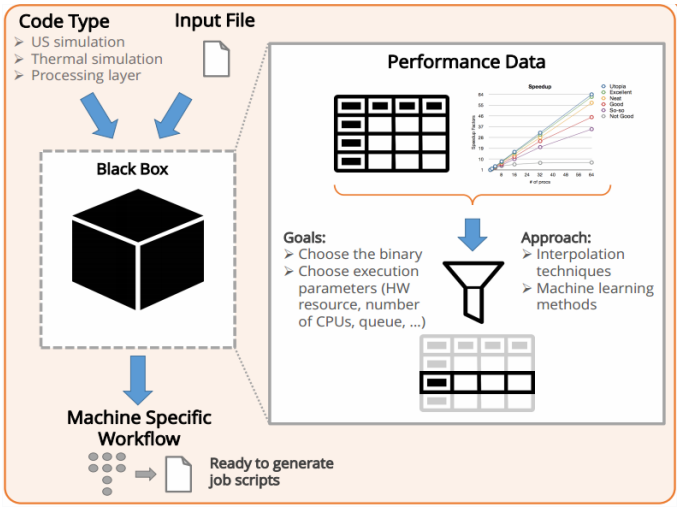


Figure 3: Optimization of the execution parameter for a given task using a couple of heuristics and historically collected performance data for known code types.

3.2 Interpolation Heuristics

The goal of interpolation heuristics is to estimate the execution time and cost using the measured performance data from previous runs. Since the users are not limited in the size of the simulation domain and many other simulation parameters influencing the execution time, the performance data will never be complete.

There are three basic situations which may happen during the execution time and cost estimation:

1. The same simulation has been seen before. In such a case, the execution time and cost can be taken as a median value over multiple records stored in the database. If the values for particular amount of resources are unknown, an interpolation is used. Figure 4a shows this situation for four different domain sizes where the performance data are only known for 1, 2, 4 and 8 threads. The values for other numbers of threads have to be interpolated, see the question marks.
2. The simulation has not been seen before. In such a case, similar simulations are sought for in the database. First, the total number of grid points is calculated as a product of the dimension sizes. This may, however, unfavourably impact the estimation, since the actual shape does have an impact on the execution time, see Section 3.3. Next, all simulations with the number of grid points close to the one being estimated are selected. Finally, the execution time and cost are interpolated from the selected data. Figure 4b shows a situation where the performance data was only measured for 4 different domain sizes. The others have to be interpolated, see the yellow area.
3. The interpolation fails and it is necessary to use queue default wall time and amount of compute resources. This may happen if the simulation is too far from the known ones, or the interpolation method begins to oscillate and produces, e.g., negative values. Fortunately, this is a transient situation because as soon as the task is executed at least once, the measured values can be used next time.

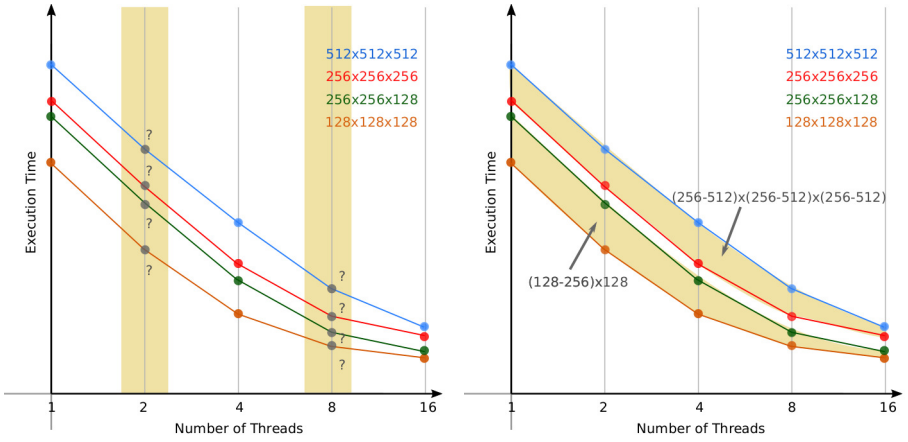
Four interpolation methods offered by the SciPy [20] Python package were investigated in this paper:

- linear interpolation (LI),
- cubic spline interpolation (CS),
- nearest neighbour interpolation (NN),
- radial basis function interpolation (RBF).

As the quality measure for the interpolation methods, *L1-*, *L2-* and *L-Infinity* norms were used [6]. Additionally, the mean percentage error of the obtained data series with respect to the measured values was calculated using Eq. (2).

$$meanPercentError = mean(\frac{|\mathbf{a} - \mathbf{b}|}{|\mathbf{a}|}) \times 100 \tag{2}$$

where $\mathbf{a}$ is a vector of reference data series and $\mathbf{b}$ is a vector of interpolated data series.



(a) Simulation size already seen. (b) Simulation size not seen before.

Figure 4: (a) The performance database misses data (highlighted in yellow) for some numbers of threads. The interpolation works with the corresponding strong scaling curves. (b) The performance database misses data for a range of domain sizes (highlighted by yellow areas). The interpolations works with several strong scaling curves from the close proximity.

3.3 k-Wave Workflow Properties

A typical biomedical ultrasound workflow consists of several data processing and numerical simulation tasks. Together, they form a workflow with approximately 100 tasks. Figure 5 shows an example of the neurostimulation workflow. While the pre- and post-processor tasks require only a single computing node, the aberration correction, forward planning, and thermal simulations may employ various executables to run on a single node, a single GPU, or multiple nodes.

The simulation domain size and timespan is given by the subject anatomy, transducer position, and the ultrasound frequency. Considering small animal neurostimulations, the domain sizes can be as small as $162 \times 192 \times 128$ grid points with 3,000 simulation time steps. The move towards human patients may expand the simulation domain size up to $768 \times 900 \times 600$ grid points with 16,800 simulation time steps.

Figure 6 shows the performance behaviour of the distributed MPI version of the k-Wave toolbox for the largest practical domain normalised to a single simulation time step. The execution times were measured on the Anselm supercomputer using 1 to 16 compute nodes, each of which with 16 cores and 64 GB of memory. It can be seen that the performance scaling is not perfect with the maximum speed-up of 6.5 yielding the parallel efficiency of 40%. The yellow, green and orange dots mark the ideal amount of computational resources for three different values of the α parameters. If the execution time is preferred, the highest possible number of nodes is selected. On the other hand, if the execution

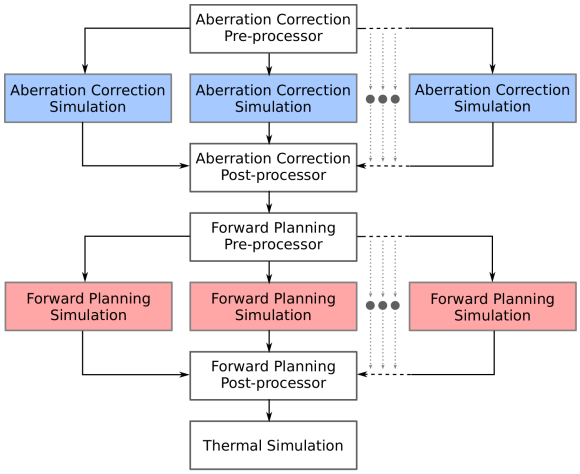


Figure 5: A neurostimulation workflow consisting of several data processing and simulation tasks. The task dependencies are shown by the arrows, meaning the simulations depicted in red or blue may be executed concurrently.

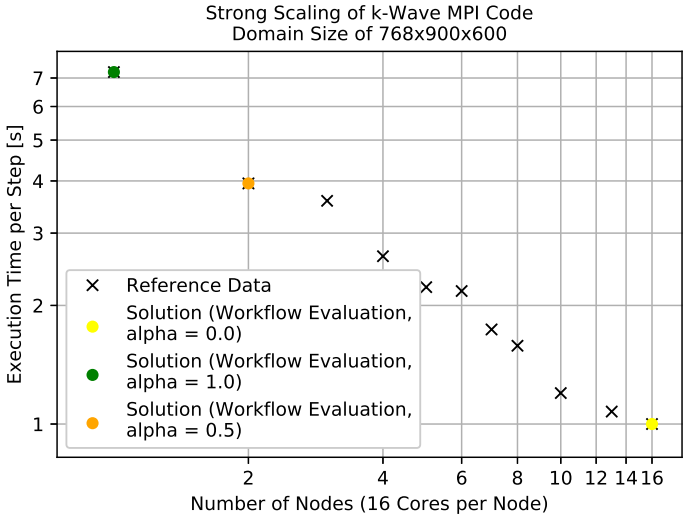


Figure 6: Strong scaling of the MPI version of the k-Wave simulation in a domain consisting of $768 \times 900 \times 600$ grid points. The yellow, green and orange dots show the best number of nodes when minimizing the computational time, computational cost, or composite workflow evaluation, respectively.

cost is preferred, a single node is selected. Finally, if both the time and cost have the same weight, two computing nodes looks as a good compromise.

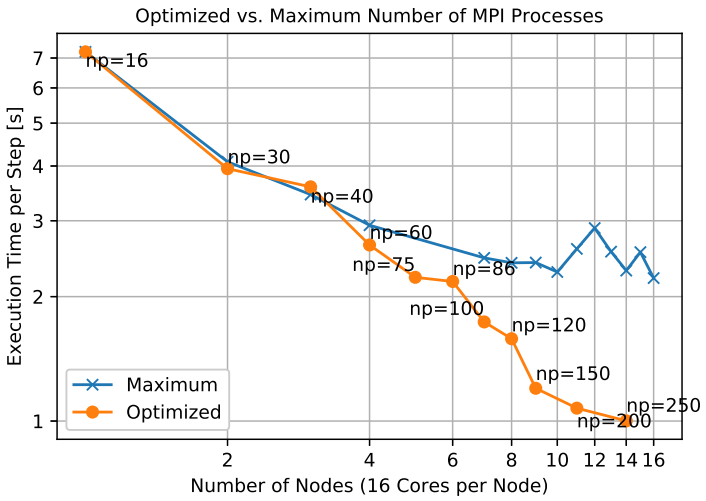


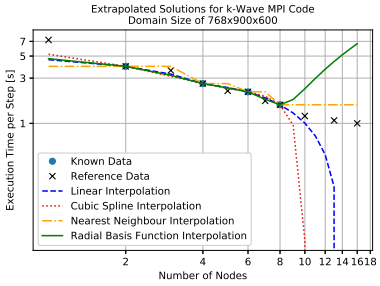
Figure 7: Strong scaling of the MPI version of the k-Wave simulation on a domain of 768×900×600 domain size executed with the maximum (blue line) and optimal (orange line) numbers of MPI processes (np).

When working with the MPI version of the k-Wave toolbox, balanced work distribution must be paid attention to. Since the code uses a one-dimensional grid decomposition over the z dimension, and the grid is z-y transposed several times every time step, the z and y dimensions must be divisible by the number of MPI processes. Otherwise, the work is not balanced evenly and the code does not scale well. Figure 7 shows the scaling of the code executed with the maximum numbers of MPI processes for given number of nodes, and with reduced numbers of processes ensuring commensurability. It is obvious, the optimized numbers of processes yield higher performance.

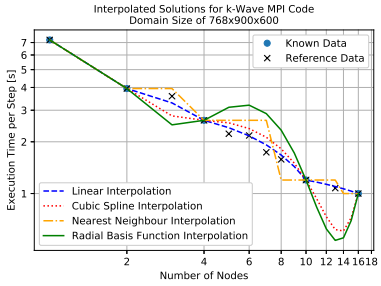
3.4 Typical Problems of Performance Data Interpolations

The interpolation and extrapolation methods have several drawbacks that will be discussed in this section. We used measured performance data from Fig. 6 and tried to manually fit interpolation curves through the measured data.

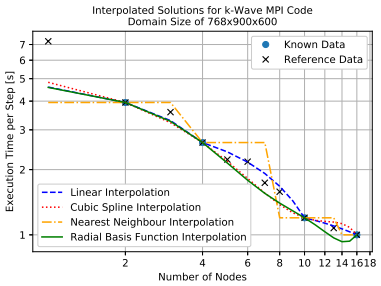
Generally, the k-Wave codes have a linearithmic computation complexity $O(n \cdot log \cdot n)$ due to extensive use of 3D Fourier transform. However, the significant amount of communication stemming from the distributed FFT may lead to quadratic communication complexity. Moreover, the proper workload balancing as well as other restrictions imposed on the domain size make the scaling even more difficult to predict [8,5]. Therefore, there are significant differences in the course of the scaling curves at low and high numbers of threads/nodes.



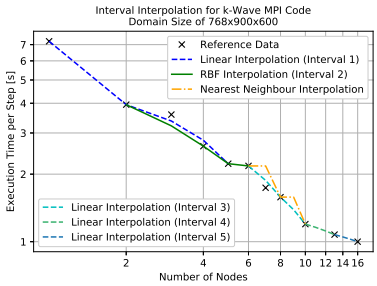
(a) Extrapolation based on the knowledge of scaling on 2, 4, 6 and 8 nodes.



(b) Interpolation based on the knowledge of scaling on 1, 2, 4, 10 and 16 nodes.



(c) Interpolation based on the knowledge of scaling on 2, 4, 10 and 16 nodes.



(d) Interval based interpolation, each interpolation uses 2 or 3 closest values.

Figure 8: (a) Unsuccessful extrapolation trained on a small number of nodes. (b) and (c) Oscillation caused by distant known values. (d) Interval interpolation not suffering from the oscillations.

Figure 8a shows a poor attempt to extrapolation where the performance data is only known for 2, 4, 6 and 8 nodes. The estimation of the execution time for number of nodes above 8 is not acceptable. The linear extrapolation as well as cubic spline extrapolation predict much shorter execution time. Nevertheless, the code scales much worse for higher number of nodes because the communication component starts to dominate. The nearest neighbour extrapolation could be used as the worst case, however, Fig. 6 suggests that the performance can even deteriorate with higher number of compute nodes. Finally, the radial basis interpolation does not produce meaningful predictions.

Figure 8b and 8c point out the need to abide appropriate interval between known values to eliminate oscillations. Figure 8b uses an additional value for one node compared to Fig. 8c. This value is usually an outlier causing unintended oscillations since having no communication. To reduce them, several interpolations may be performed on smaller intervals. The impact of this technique is shown in Fig. 8d, where the scaling data is divided into 5 intervals of 2 to 3 values. However, it is not clear how to determine the interval size automatically.

4 Experimental Results

This section describes performed experiments and the results. The experiments show the application of the selected interpolation methods in order to autonomously find the suitable execution parameters.

Due to the necessity of collecting an extensive performance dataset, we limited ourselves to only consider the OpenMP k-Wave implementation running on a single node, however, with various numbers of threads. The execution cost was then calculated as a product of the execution time and the number of processor cores used. In principle, similar results are expected to be obtained for the CUDA implementation of k-Wave. On the other hand, the MPI version poses more restrictions and may feature different results, see Sec. 3.4.

The performance data collected for the OpenMP code was obtained on Anselm with 16 cores per node, and Salomon with 24 cores per node. The performance data was divided into the training and testing datasets both of which containing over 6,500 records of the aberration correction k-Wave simulation running over 24 different domain sizes (32^3 to 512^3 grid points) and with various number of threads.

4.1 Comparison of Interpolation Techniques for Known Simulation

We first investigated the behaviour of all four interpolation techniques on the known domain size of 512^3 grid points. The first experiment used 6 known execution times from the Anselm cluster measured for 1, 2, 4, 8, 12 and 16 threads. Table 1 and Fig. 9 show the course of the interpolation functions. It can be seen that the linear and cubic spline interpolation methods reached less than 3% mean error. The linear interpolation can be thought of as a pessimistic one since overestimating the execution times. Although this may lead to a bit longer queuing times, it is safer than underestimation produced by the cubic spline interpolation, which may lead to premature termination of the simulation. The nearest neighbour interpolation shows significantly worse accuracy as well as the radial basis function interpolation deeply oscillating, especially for high numbers of threads.

The second experiment extended the number of measured values and also included the Salomon cluster. For Anselm, the performance data was extracted from the database for 1, 2, 4, 5, 8, 10, 13, and 15 threads, while for Salomon

Table 1: Comparison of selected interpolation methods for domain size of 512^3 grid points domain size and 6 known values measured on Anselm.

Interpolation Method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean Error [%]
Linear	1.27	0.59	0.46	2.89
Cubic Spline	0.93	0.42	0.35	2.29
Nearest Neighbour	4.60	2.40	2.06	9.85
Radial Basis Function	3.41	1.37	0.79	8.77

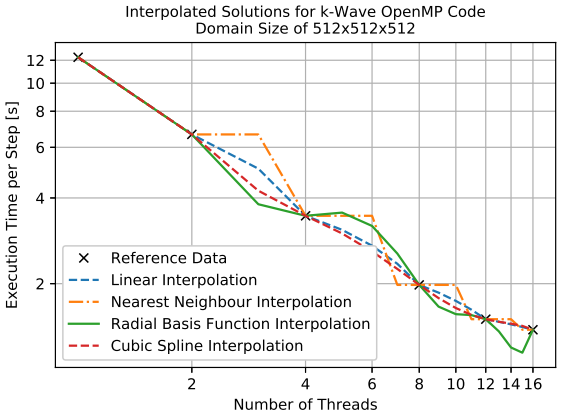


Figure 9: Comparison of various interpolation techniques for the OpenMP implementation of k-Wave running on Anselm with a domain size of 512^3 grid points.

the set was further extended by performance data for 17, 20, 22, and 24 threads. This covers 50% of all possible thread numbers usable on both clusters. The domain size remained the same (512^3 grid points).

Tables 2 and 3 show significant improvement in the prediction accuracy. The mean error produced by the linear interpolation was reduced from 2.89% to 1.81%, and 1.27% on Anselm and Salomon, respectively. Even better results were achieved for the cubic spline interpolation which produced estimation with only 1.23% and 1.12% error. Even the other interpolation methods improved

Table 2: Comparison of selected interpolation methods for domain size of 512^3 grid points domain size and 8 known values measured on Anselm.

Interpolation Method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean Error [%]
Linear	0.80	0.45	0.41	1.81
Cubic Spline	0.56	0.38	0.37	1.23
Nearest Neighbour	2.99	2.03	1.95	5.70
Radial Basis Function	1.61	0.90	0.67	4.67

Table 3: Comparison of selected interpolation methods for domain size of 512^3 grid points domain size and 12 known values measured on Salomon.

Interpolation Method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean Error [%]
Linear	0.62	0.33	0.29	1.27
Cubic Spline	0.60	0.40	0.39	1.12
Nearest Neighbour	2.73	1.71	1.63	4.68
Radial Basis Function	1.08	0.69	0.66	2.01

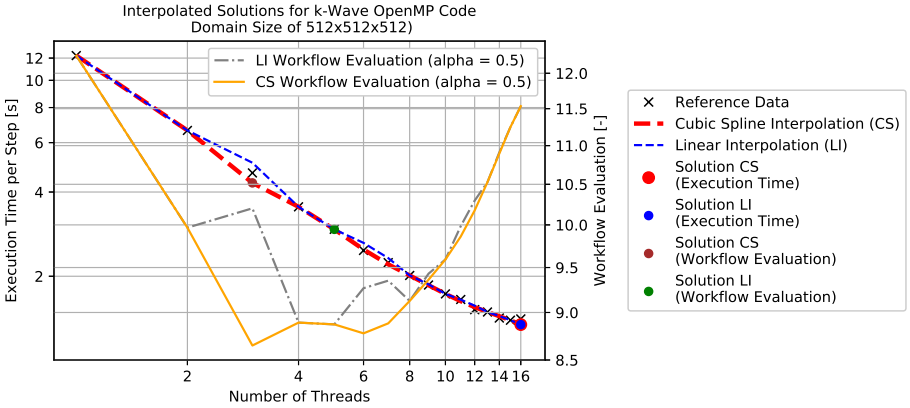


Figure 10: Estimation of the best execution configuration according to the workflow evaluation function for a domain size of 512^3 grid points on the Anselm cluster produced by linear and cubic spline interpolation.

the error close to or below 5%. This can be considered as a very good result since there is always a slight variation in execution times between different runs caused by the underlying cluster workload (mainly network and I/O parts), and variations in clock frequency amongst different cluster nodes.

Figure 10 illustrates the result of the interpolation for linear and cubic spline interpolation for the extended training set, and the domain size of 512^3 grid points. The curves show a very good agreement without any significant oscillations. The orange and grey curves are the visualizations of the workflow evaluation functions with $\alpha = 0.5$. If looking for the fastest solution, both the linear and cubic spline interpolations predict 16 threads to be the best solution. In the case the combined workflow evaluation metric is minimized, 3 and 5 threads are predicted as best compromises by the cubic spline and linear interpolations, respectively.

4.2 Comparison of Interpolation Techniques for Unknown Simulations

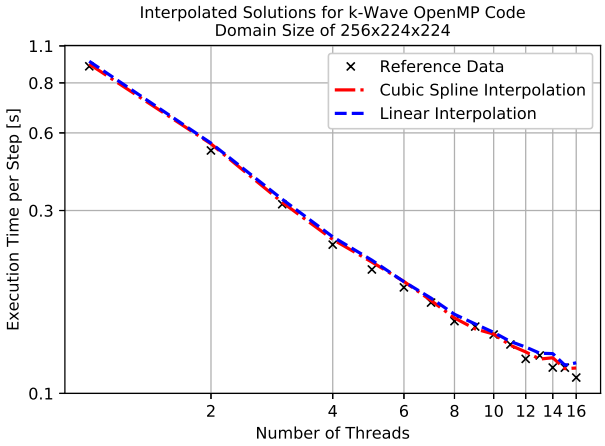
This set of experiments evaluates the capabilities of the proposed interpolation methods to estimate the execution time for simulations that have not been seen before. In this case, the closest simulations in terms of the total number of grid points are used to fit the interpolation curves. Since the results were similar for both clusters, we only present measurements on Anselm.

Three different unknown domain sizes were tested:

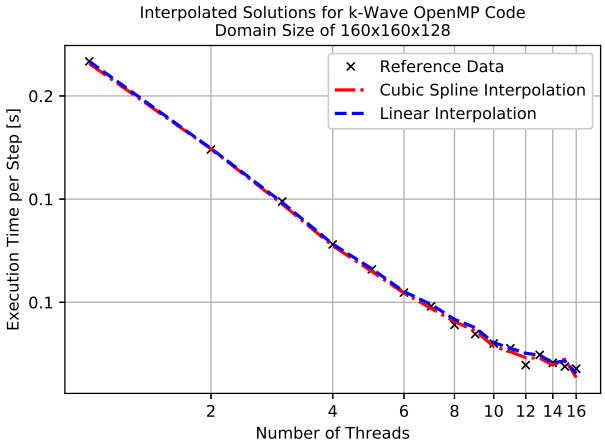
1. Tested simulation size 256×224^2, training set containing simulations of 224^3, $256^2 \times 224$, and $224^2 \times 192$ grid points.
2. Tested simulation size $160^2 \times 128$, training set containing simulations of 144^3, 160^3, and 132×128^2 grid points.

3. Tested simulation size 144^3, training set containing simulations of 160^3, 160×128^2, and 132×128^2 grid points.

Figure 11 shows the results of selected interpolations on the first two simulation domains. Both linear and cubic spline interpolations show a very close agreement with the reference data stored in the testing set. As Tables 4 and 5 quantify, the mean error for the biggest domain reaches 4.7% and 3.1% for linear and cubic spline interpolations, respectively. For the smaller domain, the error decreases to 1.75% and 2.25%. Interestingly, the cubic spline produces slightly



(a) Domain size of 256×224^2 grid points



(b) Domain size of $160^2 \times 128$ grid points

Figure 11: Comparisons of linear and cubic spline interpolation methods for unknown domain sizes. The reference data points are used for the error evaluation.

Table 4: Comparisons of selected interpolation methods for an unknown domain sizes of 256×224^2 grid points.

Interpolation Method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean Error [%]
Linear	0.17	0.056	0.034	4.724
Cubic Spline	0.11	0.037	0.025	3.073
Nearest Neighbour	0.84	0.271	0.191	22.35
Radial Basis Function	352	99.26	47.99	11492

Table 5: Comparisons of selected interpolation methods for an unknown domain sizes of $160^2 \times 128$ grid points.

Interpolation Method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean Error [%]
Linear	0.015	0.005	0.003	1.75
Cubic Spline	0.023	0.007	0.005	2.25
Nearest Neighbour	0.252	0.089	0.068	17.7
Radial Basis Function	0.371	0.121	0.089	29.2

worse estimations here. The nearest neighbour interpolation gives much worse estimation with a mean error of 22% and 18% for those two cases. Finally, the radial basis interpolation appears to be unusable for the largest domain. The extreme error is caused by high oscillations. In case of the medium-sized domain, the error decreases to 29%. Unfortunately, this still exceeds acceptable values.

The smallest domain size of interest suffers from very poor results which are summarized in Table 6 and Fig. 12. The only usable estimations are provided by the linear interpolation, however, with a mean error of 16%. The cubic spline completely fails in this case while the best estimation is surprising provided by the nearest neighbour interpolation. The radial basis interpolation also fails on this domain size. The overestimation is very likely caused by a small domain size when a single grid can fit into L3 cache memory leading to much faster execution of the Fourier transforms and overall algorithm speed-up. On the other hand, even overestimation by 200% may be thought of as acceptable considering such a simulation is executed within 2 minutes using 16 threads.

Table 6: Comparisons of linear and cubic spline interpolation methods for an unknown domain sizes of 144^3 grid points.

Interpolation Method	L1-Norm	L2-Norm	L2-Infinity Norm	Mean Difference [%]
Linear	0.196	0.061	0.041	15.99
Cubic Spline	2.080	0.527	0.185	212.6
Nearest Neighbour	0.177	0.064	0.050	13.40
Radial Basis Function	4.050	1.024	0.356	416.8

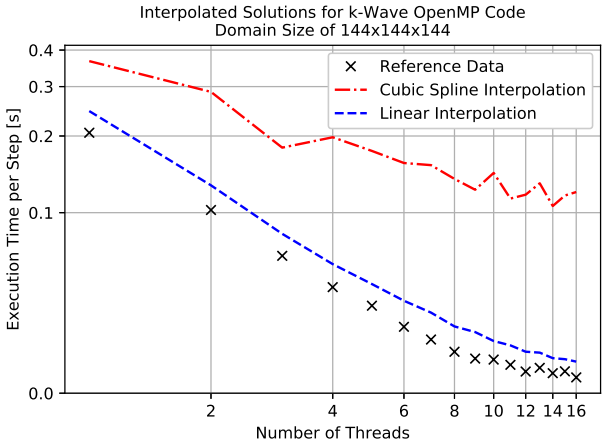


Figure 12: Comparisons of linear and cubic spline interpolation methods for unknown domain size of 144^3. The reference data points are used for the error evaluation.

5 Conclusions

The need for offloading complex scientific workflows to cluster and cloud environment is ubiquitous. k-Dispatch is a workflow management system providing automated execution, planning and monitoring of biomedical workflows composed of k-Wave ultrasound and thermal simulations. Its interface enables connection of various user applications and unifies the access to different computational resources.

One of the key challenges in automated execution of complex workflows is the proper setting of execution parameters for particular tasks. Since the end users have no or very limited knowledge about the amount of computational resources to be allocated for each task, it is necessary to provide as good estimation as possible based on the performance characteristics of particular codes and actual input data. Unsuitable values may lead to long queueing times or early tasks termination due to exhausted time allocation.

This paper has presented a single pass algorithm traversing the workflow and optimizing the execution parameters for every task independently. For every task, the input file is inspected, the task parameters retrieved, and the performance database searched for similar ones. If there is a direct match, the execution time and cost are loaded for known execution parameters, i.e., number of compute nodes, GPUs, processor cores, etc. Missing values may be filled in using interpolation techniques. However, if the task parameters have not been seen before, the interpolation is used to estimate the execution time and cost using a training set composed of tasks with similar parameters.

Four different interpolation techniques have been investigated. When the task parameters have been seen before, the cubic spline interpolation showed

the best results with mean error between 1.12% and 2.29%. In the case the task parameters have not been seen before, the linear interpolation showed the best results. Depending on the similarity of the records found in the performance database, the mean error varies between 1.17% and 15%. It should be noted that the highest error showed up only for very small tasks where the overestimation of execution time or cost do not play a significant role.

5.1 Future Work

Future work will be focused on multi-pass optimization of workflow execution parameters. The goal is to minimize not only the execution time and cost but also the queuing times. This however requires the knowledge of the actual cluster workload and queues occupancy as well as a cluster simulator to quickly estimate the queuing times for the whole workflow under different execution parameters. We are considering the adaptation of the ALEA simulator [11] to match the scheduling algorithms and hardware configurations of IT4Innovations clusters, and the characteristics of the k-Wave workflows.

We would also like to implement more sophisticated heuristics to select an appropriate number of compute nodes as well as optimal number of MPI processes for large simulations to avoid performance penalizations. Consequently, we would like to study machine learning methods since we expect to have collected large performance dataset, and perform experiments on both, artificial and real-world workflows.

6 Acknowledgement

This work was supported by the FIT-S-17-3994 Advanced parallel and embedded computer systems project. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602 and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center - LM2015070. This work was supported by the Engineering and Physical Sciences Research Council, United Kingdom, grant numbers EP/L020262/1, EP/M011119/1, EP/P008860/1, and EP/S026371/1.

References

1. Abbas, A., Coussios, C., Cleveland, R.: Patient specific simulation of hifu kidney tumour ablation. vol. 2018, pp. 5709–5712 (07 2018). https://doi.org/10.1109/EMBC.2018.8513647
2. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming **13**, 219–237 (2005)

3. Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., da Silva, R.F., Livny, M., Wenger, K.: Pegasus: a workflow management system for science automation. Future Generation Computer Systems (2014)
4. Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. Journal of Computer Science and Technology **21**(4), 513–520 (jul 2006). https://doi.org/10.1007/s11390-006-0513-y
5. Frigo, M., Johnson, S.: The Design and Implementation of FFTW3. Proceedings of the IEEE **93**(2), 216–231 (feb 2005). https://doi.org/10.1109/JPROC.2004.840301
6. Gradshtein, I.S.: Table of integrals, series, and products. Academic Press, San Diego (2000)
7. Grisey, A., Yon, S., Letort, V., Lafitte, P.: Simulation of high-intensity focused ultrasound lesions in presence of boiling. Journal of Therapeutic Ultrasound (2016). https://doi.org/10.1186/S40349-016-0056-9
8. Jaros, J., Rendell, A.P., Treeby, B.E.: Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. The International Journal of High Performance Computing Applications **30**(2), 137–155 (may 2016). https://doi.org/10.1177/1094342015581024
9. Jaros, M., Treeby, B.E., Georgiou, P., Jaros, J.: k-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources. In: Proceedings of the Platform for Advanced Scientific Computing Conference. PASC 20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3394277.3401854
10. Kadlubiak, K., Jaros, J., Treeby, B.E.: GPU-Accelerated Simulation of Elastic Wave Propagation. In: 2018 International Conference on High Performance Computing & Simulation (HPCS). pp. 188–195. IEEE (jul 2018). https://doi.org/10.1109/HPCS.2018.00044
11. Klusacek, D., Toth, S., Podolnikova, G.: Complex Job Scheduling Simulations with Alea 4. CEUR Workshop Proceedings **1828**, 53–59 (2017). https://doi.org/10.1145/1235
12. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience **18**(10), 1039–1065 (aug 2006). https://doi.org/10.1002/cpe.994
13. Manohar, S., Dantuma, M.: Current and future trends in photoacoustic breast imaging. Photoacoustics **16** (12 2019). https://doi.org/10.1016/j.pacs.2019.04.004
14. Mohammadi, L., Behnam, H., Tavakkoli, J., Avanaki, M.R.: Skull's photoacoustic attenuation and dispersion modeling with deterministic ray-tracing: Towards real-time aberration correction. Sensors (Switzerland) (2019). https://doi.org/10.3390/s19020345
15. Robert, Y.: Task Graph Scheduling, pp. 2013–2025. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-0-387-09766-4_42
16. Suomi, V., Jaros, J., Treeby, B., Cleveland, R.: Nonlinear 3-D simulation of high-intensity focused ultrasound therapy in the Kidney. pp. 5648–5651. IEEE (aug 2016). https://doi.org/10.1109/EMBC.2016.7592008
17. Suomi, V., Treeby, B., Jaros, J., Makela, P., Anttinen, M., Saunavaara, J., Sainio, T., Kiviniemi, A., Blanco, R.: Transurethral ultrasound therapy of the prostate in the presence of calcifications: A simulation study. Medical physics **45** (09 2018). https://doi.org/10.1002/mp.13183

18. Treeby, B.E., Cox, B.T.: K-wave: Matlab toolbox for the simulation and reconstruction of photoacoustic wave-fields. Journal of Biomedical Optics **15**(2), 021314 (2010)
19. Treeby, B.E., Jaros, J., Rendell, A.P., Cox, B.T.: Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method. The Journal of the Acoustical Society of America **131**(6), 4324–36 (2012). https://doi.org/10.1121/1.4712021
20. Virtanen, P., Gommers, R., Oliphant, T.E., et.al: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2