Provenance-integrated parameter selection and optimization in numerical simulations

Julia Kühnert
IANS/IPVS - University of Stuttgart

Dominik Göddeke
IANS - University of Stuttgart

Melanie Herschel
IPVS - University of Stuttgart

Abstract

Simulations based on partial differential equations (PDEs) are used in a large variety of scenarios, that each come with varying requirements, e.g., in terms of runtime or accuracy. Different numerical approaches to approximate exact solutions exist, that typically contain a multitude of parameters that can be tailored to the problem at hand. We explore how high-level provenance, i.e., provenance that is expensive to capture in a single simulation, can be used to optimize such parameters in future simulations for sufficiently similar problems. Our experiments on one of the key building blocks of PDE simulations underline the potential of this approach.

1 Introduction

Many important application problems are modeled mathematically by *partial differential equations (PDEs)*, i.e., sets of equations describing the interaction and evolution of quantities and their derivatives. A prominent model problem is the diffusion equation $\partial_t u + \Delta u = f$ with the Laplace operator $\Delta = \sum_{i=1}^{d} \partial_i^2$, which can be found as a submodel in, e.g., fluid dynamics, structural mechanics, but also in the social sciences. In general, solutions of PDEs cannot be expressed explicitly as formulae, and hence numerical schemes must be employed to express the problem at hand in computer-tractable form and to compute (approximate) solutions.

Different applications typically lead to different requirements on the accuracy, memory consumption, and computation time of the simulation. In some cases, accuracy is of utmost importance, for instance in safety-critical scenarios. In other cases, runtime is the limiting resource, e.g., in weather prediction or when real-time constraints have to be met.

The idea to log meta-data about executed programs, including simulations, for various purposes is not new [3, 8]. In particular, this holds for hardware-related data such as runtime and memory consumption, which is typically used to allocate resources for follow-up simulations with different model parameters. In this paper, we focus on meta-data relating to the numerical schemes that are used in the simulation. Here, logging of key indicators is also widespread, e.g., the number of iterations or the temporal evolution of error indicators to monitor the progression towards the solution. We refer to such meta-data that can be logged alongside the regular processing as *low-level provenance*.

In general, alternative numerical schemes to solve a PDE model, as well as schemes for sub-problems such as the solution of linear equation systems, have certain characteristics that directly or indirectly determine the computational cost. The common practice today of choosing a suitable numerical method for the problem at hand and setting the parameters of the method in a way that the solution can be computed efficiently is based on user experience. Incorporating meta-data that quantify the behavior and performance of different schemes under varying conditions to ultimately automate the selection and parameterization process has received only little attention. We use the term performance in a wide sense here, encompassing runtime, accuracy, but also applicability. Obtaining the desired meta-data can itself be quite costly, possibly necessitating expensive extra simulations for parameter settings that are not covered before. We differentiate this new kind of provenance from the previously mentioned loggable provenance by calling it *high-level provenance*.

Our long-term goal is to feed provenance to a simulation optimizer that determines a good execution strategy for a simulation. This resembles the optimization of database queries, where an optimizer determines a physical query execution plan for a declarative query by weighing different alternatives for execution of the query against each other based on a cost model. Similar to, e.g., [5, 7], the provenance is considered in defining and adapting the cost model. However, in our setting, the optimizer additionally has to weigh the cost of computing high-level provenance against its potential benefit.

Contributions. We introduce the novel idea of using high-level provenance for automated scheme selection and parameterization (Section 2). For a specific model problem, we further introduce a first metric for high-level provenance (Section 3). Our preliminary evaluation showcases that we can rely on this metric to select parameters resulting in good

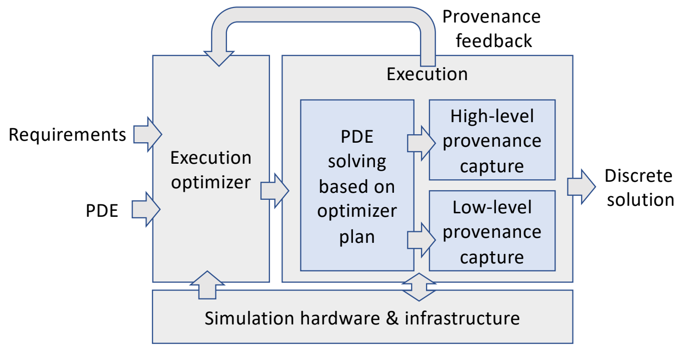

Provenance feedback
Execution
Requirements
PDE
Execution optimizer
PDE solving based on optimizer plan
High-level provenance capture
Low-level provenance capture
Discrete solution
Simulation hardware & infrastructure

Figure 1: Overview of our general approach

performance for similar problems (Section 4).

2 General framework

Figure 1 depicts our general framework for automated scheme selection and parameterization. It takes as input a PDE, possible performance requirements, and hardware characteristics.

Based on the input, the **optimizer** decides on the best way to solve the PDE. The optimizer is intended to support or even substitute experts. To allow for informed decisions, it ingests provenance from simulation runs, which are analyzed to determine suited execution plans for future similar problems.

The **execution** component in Figure 1, with its subcomponents highlighted in blue, is at the core of this paper. To **solve** the PDE, a specific scheme needs to be applied. We focus on a common pattern such schemes follow: first transform the PDE, possibly after linearization and applying a time-stepping scheme, to a (series of) linear equation system(s) $Ax = b$. The matrix A is large and sparse, which is then exploited by iterative solvers. After reaching a certain tolerance the computation stops with an approximate solution, resulting in less computation compared to direct solvers. To further improve iterative solvers, preconditioning linear equation systems can be used. Even when following this pattern, there are numerous iterative solvers, preconditioners, or parameterizations to consider. This motivates the collection of provenance of different computation steps during execution, such that their performance characteristics can be derived.

We first consider **low-level provenance**. It comprises metadata about the execution that is cheap to determine, as it can be simply logged alongside computations, e.g., general performance and accuracy metrics and metrics characterizing specific numerical schemes. In addition, we introduce the notion and use of **high-level provenance** that is based on characteristics and new metrics for PDEs, linear equation systems, and other computation steps, that we use to determine a model for their varying cost and behavior. To make an analogy to "classical" data provenance research [1], low-level provenance resembles why-provenance as it captures performance metrics that contribute to the overall performance – why-provenance captures which tuples in a database contribute to the result of a database query. Then, high-level provenance can be seen as an analogy to how-provenance, as it reflects how the metrics captured by low-level provenance impact the overall performance – in databases, how-provenance tells how tuples part of why-provenance combine to yield result tuples. The metrics to be identified for high-level provenance are highly specific for a chosen scheme, therefore, Section 3 focuses on high-level provenance for a specific model problem. Also, they are not necessarily by-products of the ordinary simulation and thus are potentially expensive to compute. Consequently, deciding to collect high-level provenance and scheduling this computation is an interesting research problem by itself.

3 High-level provenance for a model problem

As a first proof of concept for the outlined approach, we consider a specific model PDE problem, and differentiate between low-level *scheme parameters* and their influence on the behavior of the scheme to determine good parameters for runs of the PDE problem with different, previously not sampled high-level *model parameters*.

Model problem. On a domain $\Omega =]0,1[^2$, we consider

$$-\alpha\partial_x^2 u - \beta\partial_y^2 u = f \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega$$

with model parameters $\alpha, \beta \in \mathbb{R}$. Such anisotropic Poisson problems are prototypical for many elliptic PDEs and can be found, as a sub-model, in many applications (cf. Section 1). α and β determine the degree of anisotropy in the system. For $\alpha = \beta$, we have a homogeneous material in which the modeled stationary diffusion process takes place, while for $\beta \neq \alpha$, the material has preferred directions, e.g., slate.

Discretization. Following the execution component in Section 2, we cover Ω with an equidistant grid of spacing $h > 0$, and apply second order centered finite differences. This results in a linear system $Av = b$ for the discrete unknown values v_i that approximate the unknown continuous solution u in the grid points. We choose powers of 1/2 for h, so that A quickly becomes large. Also, A is symmetric, positive definite and sparse, with only five entries per row independent of h. The nonzero pattern of A is independent of the choice of α, β and h, just the values of the nonzero matrix entries change. b is chosen as Au for $u(x,y) = (x(1-x)y(1-y))^2$.

Solver. Due to the properties of A, the conjugate gradient method (CG, [4]) is a favorable scheme to solve the linear system. CG is an iterative method: in each iteration the dimension of a basis of the solution space is increased by one through a clever projection, and an approximate solution is formed in that basis. We denote the k-th iterate as $v^{(k)}$. For the error $e^{(k)} = v^{(k)} - A^{-1}b$, the bound $\|e^{(k)}\|_A \leq 2\left(\frac{\sqrt{\mathrm{cond}_2(A)}-1}{\sqrt{\mathrm{cond}_2(A)}+1}\right)^k \|e^{(0)}\|_A$ holds in the energy norm, with the condition number $\mathrm{cond}_2(A) = \|A\|_2\|A^{-1}\|_2$. The latter is a measure for the sensitivity of A with respect to small variations in the data, and due to the bound, also a performance indicator in terms of convergence speed. Computing A^{-1} is even more expensive than solving the original system, as in general, A^{-1} is much denser than A, i.e., it has much more nonzero entries. For our model problem, the condition number depends on the quotient of the model parameters α and β. We thus obtain a computable characteristic of the model parameters for high-level provenance without computing $\text{cond}_2(A)$.

Preconditioning. To counteract the influence of the condition number on the convergence speed, preconditioning can be applied in each iteration of the CG algorithm, so that it effectively solves the system $PAv = Pb$. Obviously, for $P \approx A^{-1}$, the system becomes trivial, but computing A^{-1} is infeasible. We use the *dual threshold incomplete LU preconditioner (ILUT)* introduced by Saad [6], which approximates the lower and upper triangular factors $\tilde{L} \approx L$ and $\tilde{U} \approx U$ of the LU factorization $A = LU$. We can write $A = \tilde{L}\tilde{U} + F$ for some remainder matrix F that is not stored. Then, in each iteration of the CG algorithm, two auxiliary triangular systems have to be solved, which is easily achieved by forward and backward substitution. Also, a *dropping strategy* determines which entries are used for the incomplete LU factorization, and which entries are ignored, i.e., placed virtually in F. In the ILUT preconditioner the dropping strategy depends on two parameters, the drop-tolerance τ and the fill-in p. In a first step the drop-tolerance τ is multiplied with the average magnitude of the current row i of A, resulting in the relative drop-tolerance τ_i. During the factorization all elements that are smaller than this relative drop-tolerance τ_i are replaced by zero. In a second step only the p largest elements in the row of L and U are kept in addition to the diagonal element. A small drop-tolerance τ and a big fill-in p therefore result in a high amount of nonzero entries in the preconditioner, and consequently high costs to compute it a priori, and to apply it in each CG iteration. In our experiments, we use the ILUT implementation from the Eigen library [2], which does not use p, but rather a fill-factor, which relates to the fill-in p by $p = \frac{\text{nonzeros}(A)\cdot\text{fill-factor}}{n} + 1$.

As τ and p influence $\tilde{L}$ and $\tilde{U}$ in a highly nonlinear fashion, it is in general impossible to determine these *scheme parameters* a priori. Bad choices may lead to too dense and thus too expensive factors, even though the CG solver would then converge rapidly. It may also happen that the preconditioned system PA is no longer symmetric positive definite, causing CG to fail. An optimized selection of the parameters is therefore essential, and typically involves careful balancing of all related costs, and the improvement of the convergence speed.

Provenance. We log the model parameters α and β, the scheme parameters τ and fill-factor, as well as the number of needed iterations, the solution time of the CG method, the assembly time of the ILUT preconditioner, the number of nonzeros in the ILUT preconditioner, the tolerance, and others. To detect solver failure, we prescribe a maximum number of iterations, θ_{it}. In successful configurations, the CG solver terminates upon reaching an absolute tolerance of θ_{tol}. The anisotropy parameters serve as a high-level metric on the difficulty of the problem. Together with the other low-level data we can predict the simulation performance including the required computation time. The number of nonzeros in the ILUT preconditioner give us a metric on the preconditioner storage and application costs. With the parameters for the ILUT preconditioner we can adapt the computation in time, accuracy and storage.

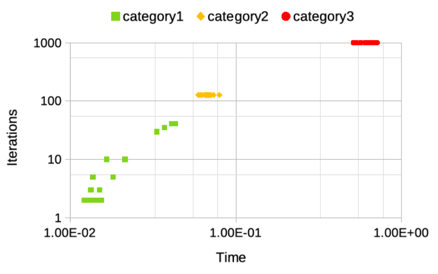
4 Experimental evaluation

We fix $h = 1/128$, leading to $n = 16\,129$ unknowns. We report results for $\theta_{tol} = 10^{-14}$ and $\theta_{it} = 1\,000$. They are representative of results we also obtained for different problem dimensions and θ_{tol}. The considered anisotropies, i.e., $\alpha = 1$ and $\beta \in \{10^{-7}, 10^{-6}, \ldots, 10^{7}\}$, span a wide range of applications, e.g., diffusive processes in practically relevant porous media. In our experiments, we focus on the preconditioner, and explore the performance characteristics for drop-tolerances $\tau \in \{10^{-16}, 10^{-14}, \ldots, 10^{4}\}$ and integers between 0 and 10 for the fill-factors. We do not explore all parameter combinations further that lead to more than ten nonzero entries per row in the preconditioner, to emulate memory limitations of twice the requirements for the system matrix.

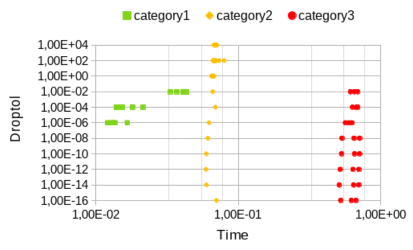
Categories of Convergence. Figure 2a shows iterations over total time (ILUT assembly plus CG solver), exemplarily for $\beta = 10^{6}$. We observe similar patterns as long as the anisotropy is large enough, i.e., $\beta \notin [10^{-1}, 10^{1}]$. Each data point in the plot corresponds to a different parameter combination, and we can identify three categories: Category 1 contains the sought 'sweet spots', while category 3 contains all cases that reached θ_{it} iterations without convergence. Clearly, having a configuration ending up in this category is undesirable, as it fails to solve the problem. In category 2, all configurations require 127 iterations, which matches the baseline when not applying preconditioning, in other words, the effort associated with constructing and applying the preconditioner is wasted. The number of nonzeros (the storage cost for the preconditioner) varies from moderate (78 000) for category 1, to comparable to a diagonal preconditioner (16 129) for category 2, to excessive (158 571) for category 3. In terms of the envisioned optimizer, we aim for category 1 and exclusion of category 3.

Patterns in the ILUT parameters. To find configurations that avoid category 3 or ideally yield category 1, we study the influence of ILUT parameters in more detail. Figure 2b shows that τ is not suitable for separating the identified categories. In contrast, Figure 2c shows that the fill-factors allow to distinguish categories 1 and 3, meaning that we can avoid non-converging parameter settings by excluding fill-factors below 3, while the optimum in this set of experiments is reached for a fill-factor of 9.

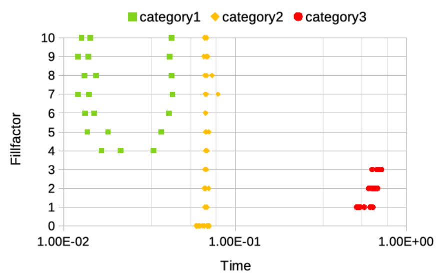
ILUT parameters for a new problem. While the previous



(a) Iterations w.r.t. time



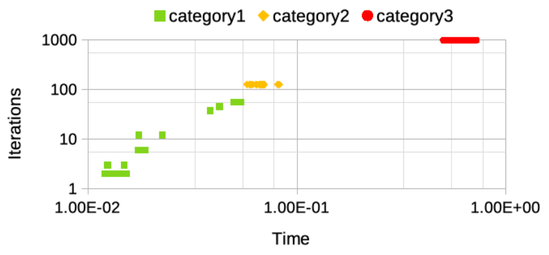
(b) Drop-tolerances w.r.t. time



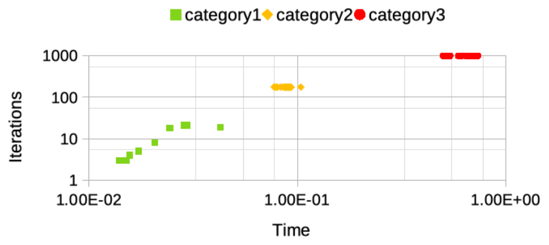
(c) Fill-factors w.r.t. time

Figure 2: Iterations (a), drop-tolerances (b), and fill-factors with respect to time when setting $\beta = 10^6$

experiments provided indications for the optimizer to avoid category 3, we now evaluate if a set of 'good' parameters for a certain degree of anisotropy is also beneficial for similar anisotropies, i.e., for problem instances that have not been simulated before. Figure 3 illustrates the results for $\beta = 10^7$ and $\beta = 10^5$, where the coloring of the parameter settings has been carried over from the previous experiments. We see that our approach of employing high-level provenance works remarkably well, as the three categories identified for $\beta = 10^6$ are the same for $\beta = 10^5$ and $\beta = 10^7$, and we can immediately start with an arbitrary choice from the first category. A more detailed analysis (not shown) reveals that the optima are different for different β values, but this only affects the ordering within each category. Also, for $\beta = 10^5$, more parameter configurations of category 1 were pruned by our limit on the storage cost for the preconditioner. The transgression is minor and thus those parameters would also be suitable.



(a) Iterations w.r.t. time for $\beta = 10^7$



(b) Iterations w.r.t. time for $\beta = 10^5$

Figure 3: Iterations plotted over time for β values in the neighborhood of $\beta = 10^6$. Categories colored according to $\beta = 10^6$

5 Summary and Future work

We found an approach to the parameter selection for the ILUT preconditioner based on the level of anisotropy. The acquisition of knowledge requires so far a complete simulation for every parameter and is therefore not practical. We want to further investigate the idea of computing only a few iteration steps to distinguish between the categories. The next steps include modifications of our model problem with regards to different boundary values and an extension to different iterative solvers. Afterwards, we will focus on the simulation optimizer that steers high-level provenance computation and determines a good execution based on the gathered data. Finally, to devise a more general solution, further investigation on metrics and their combination for a variety of differential equations is necessary to determine which high-level provenance to collect.

Acknowlegements. Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2075 – 390740016.

References

[1] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends Databases*, 1(4):379–474, 2009.

[2] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[3] M. Herschel, R. Diestelkämper, and H. Ben Lahmar. A survey on provenance: What for? what form? what from? *The VLDB Journal*, 26(6):881–906, 2017.

[4] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.

[5] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proceedings of the VLDB Endowment (PVLDB)*, 12(11):1705–1718, 2019.

[6] Yousef Saad. Ilut: A dual threshold incomplete lu factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994.

[7] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - db2's learning optimizer. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 19–28, 2001.

[8] Young-Kyoon Suh and Ki Yong Lee. A survey of simulation provenance systems: modeling, capturing, querying, visualization, and advanced utilization. *Human-centric Computing and Information Sciences*, 8:27, 2018.