X International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2023
M. Papadrakakis, B. Schrefler and E. Oñate (Eds)

DATA-PARALLEL RADIAL-BASIS FUNCTION INTERPOLATION IN PRECICE

DAVID SCHNEIDER[†*], TIMO P. SCHRADER[†*], AND BENJAMIN UEKERMANN[†]

[†]Institute for Parallel and Distributed Systems
University of Stuttgart
Universitätstraße 38, 70569, Stuttgart
e-mail: david.schneider@ipvs.uni-stuttgart.de, web page: https://www.ipvs.uni-stuttgart.de/
[*]Equal contributors

Key words: multiphysics coupling, data mapping, radial-basis functions, high-performance computing, GPGPU computing

Abstract. We present data-parallel approaches to solve radial-basis function interpolation problems in the context of partitioned multi-physics simulations, where interpolation methods are required to transfer coupling data between non-matching vertex clouds. Data-parallel approaches are a key component for the efficient use of accelerator cards and thus for performance portability on modern compute platforms. The presented approach is integrated into the open-source coupling library preCICE.

After discussing different implementation strategies, we introduce a solution based on the linear algebra library Ginkgo, which provides a common abstraction layer for cross-platform performance with focus on solving sparse linear systems. The new implementation exploits accelerator cards for both, matrix assembly as well as solving the resulting linear system. The capability of the presented approach is compared to already existing implementations in preCICE using a turbine blade geometry.

1 INTRODUCTION

Solving interpolation problems using radial-basis functions (RBF) is a computationally demanding task. The application area of RBF interpolation ranges from implicit surface reconstruction to solving partial differential equations [1]. In this article, we analyze efficient RBF interpolation in the context of partitioned multi-physics simulation, where interpolation methods are required to transfer coupling data between non-matching coupling meshes. In particular, we focus on data-parallel execution of RBF interpolation problems to exploit the compute capabilities of modern multi- and many-core systems, such as graphics processing units (GPU).

This article revolves around data mapping within the open-source coupling library preCICE[1] [2]. preCICE is designed for partitioned multi-physics simulations, but has no specific notion about the simulated physics. All data defining the interpolation problem is provided by the user, usually as a cloud of vertices with attached data. Therefore, a typical data-mapping

[1]`https://precice.org/`

problem in preCICE consists of one vertex cloud with user-provided data values used to build an interpolant, which we refer to as the input mesh, and one vertex cloud used to evaluate the interpolant to retrieve data, which we refer to as the output mesh. RBF interpolation, among other data mapping methods, is applied to deal with these vertex clouds in preCICE. For RBF data mappings, preCICE offers already a sequential solver algorithm as well as an MPI-parallel solver algorithm capable of massively-parallel runs on supercomputers [3]. However, none of these approaches is yet able to exploit data-parallel execution utilizing accelerator cards. Therefore, on top of these two implementations, we present the integration of a data-parallel RBF mapping implementation that specifically targets accelerator cards. We discuss the newly added data-parallel implementation approach and show when it is feasible to apply over already existing implementations. This article builds on and summarizes the master thesis of Schrader [4].

The remainder of this article is organized as follows: Section 2 describes the mathematical details of RBF interpolation, Section 3 summarizes already existing implementation approaches for RBF interpolation in preCICE, Section 4 discusses potential implementation approaches for data-parallel implementation and concludes on their applicability in preCICE, Section 5 shows the integration of our implementation approach, Section 6 presents results on numerical test cases, and Section 7 concludes.

2 RADIAL-BASIS FUNCTION INTERPOLATION

Interpolation methods in coupled simulations are required in order to map data between input and output meshes. Each mesh is defined in terms of a discrete set of vertices $\{\mathbf{x}_i \in \mathbb{R}^d; i = 1, \dots N\}$ in $d \in \{2, 3\}$ space dimensions. Given corresponding data values $\{f_i \in \mathbb{R}; i = 1, \dots N\}$ for the discrete vertices on the input side, data mapping aims to interpolate a solution at the discrete vertices of an output mesh. Oftentimes, the mesh vertices $\mathbf{x}_i$ are initially defined once in coupled simulations, while only the data values f_i change over time. RBF interpolation constructs an interpolant $\mathcal{I}_f : \mathbb{R}^d \to \mathbb{R}$ by

$$\mathcal{I}_f(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + \beta_0 + \boldsymbol{\beta}_l^T \cdot \mathbf{x}, \tag{1}$$

where N is the number of input mesh vertices, $\lambda_i \in \mathbb{R}$ represents the unknown coefficients, $\phi : \mathbb{R} \to \mathbb{R}$ a radial-basis function, and $\beta_0 \in \mathbb{R}$ and $\boldsymbol{\beta}_l \in \mathbb{R}^d$ represent the unknown coefficients of an additional linear global polynomial contribution. The global polynomial is required to fulfill consistency constraints, i.e., exactly interpolate linear data. We enforce N constraints through the interpolation condition

$$\mathcal{I}_f(\mathbf{x}_i) = f_i,\ i = 1, \dots N \tag{2}$$

and additional regularization constraints for a unique solution according to

$$\sum_{i=1}^{N} \lambda_i \cdot \mathbf{x}_i = \mathbf{0} \text{ and } \sum_{i=1}^{N} \lambda_i = 0. \tag{3}$$

Summarizing the constraints in a matrix notation leads to the following linear system

$$\begin{pmatrix} \mathbf{C} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}, \tag{4}$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_N)^T \in \mathbb{R}^N$, $\boldsymbol{\beta} = \left(\beta_0, \boldsymbol{\beta}_l^T\right)^T \in \mathbb{R}^{d+1}$, $\mathbf{C} = \left(\phi\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right)\right)_{i,j=1,\dots,N} \in \mathbb{R}^{N\times N}$, $\mathbf{P} = \left(1, \mathbf{x}_i^T\right)_{i=1,\dots,N} \in \mathbb{R}^{N\times(d+1)}$, and $\mathbf{f} = (f_1, f_2, \dots, f_N)^T \in \mathbb{R}^N$. Solving Eqn. (4) for $\boldsymbol{\lambda}$ and $\boldsymbol{\beta}$ enables the evaluation of Eqn. (1) at any given output mesh vertex $\mathbf{x}$ [5].

preCICE offers different options for ϕ. Throughout the remainder of this work, we use the compact thin-plate-spline C2 [2]

$$\phi(r) = \begin{cases} 1 - 30r^2 - 10r^3 + 45r^4 - 6r^5 - 60r^3 \ln(r) & \text{if } r < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The radial distance r between two vertices is computed by $r = \|\mathbf{x}_i - \mathbf{x}_j\|_2/\rho$ with the support radius ρ of the basis-function determining the sparsity of matrix $\mathbf{C}$ as well as the accuracy of the RBF interpolant. Here, ϕ is strictly positive definite, which results in a symmetric positive definite (s.p.d.) matrix $\mathbf{C}$. To take advantage of the sparsity of $\mathbf{C}$, we do not solve Eqn. (4) exactly [6]. Instead, we split the polynomial coefficients $\boldsymbol{\beta}$ from Eqn. (4) and solve them separately in a linear least-square system

$$\|\mathbf{P} \cdot \boldsymbol{\beta} - \mathbf{f}\|_2 \to \min \tag{6}$$

Afterwards, the basis-function coefficients $\boldsymbol{\lambda}$ can be computed solving

$$\mathbf{C} \cdot \boldsymbol{\lambda} = \mathbf{f} - \mathbf{P} \cdot \boldsymbol{\beta}. \tag{7}$$

Eqn. (6) and Eqn. (7) need to be solved for each data exchange between coupling participants, potentially hundreds of thousand times per simulation. However, matrices $\mathbf{C}$ and $\mathbf{P}$ remain constant throughout the simulation, if the vertices $\mathbf{x}_i$ remain constant. Direct solution methods can exploit this property by computing factorizations of $\mathbf{C}$ and $\mathbf{P}$ once and solving Eqn. (6) and Eqn. (7) efficiently using the factorized matrices.

3 EXISTING SERIAL AND DISTRIBUTED MEMORY IMPLEMENTATIONS

RBF interpolation is already supported in preCICE by employing two different implementation approaches: a serial implementation relying on the linear-algebra library Eigen [7] and an MPI-parallel version relying on PETSc [8].

The serial implementation applies dense matrix decompositions for matrices $\mathbf{C}$ and $\mathbf{P}$. A Cholesky decomposition is applied for matrix $\mathbf{C}$ in case the selected basis function ϕ is positive definite, otherwise a QR decomposition is used. Matrix $\mathbf{P}$ is always factorized using a QR decomposition. If preCICE runs MPI-parallel in a distributed-memory environment, the mapping computation uses a gather-scatter approach to collect the data on a single process, i.e., the mapping is still executed in serial on a single CPU.

The second implementation uses sparse data-structures and solvers from the parallel linear-algebra library PETSc [9]. Similar to the serial implementation, the tall and skinny matrix $\mathbf{P}$ of the polynomial least-square system is solved using a QR decomposition. However, the coefficient matrix $\mathbf{C}$ is solved using an iterative Krylov method. By default a GMRES solver is used, independent of the basis functions. For the experiments in this work, however, a conjugate-gradient (CG) solver is used. To report convergence, a relative convergence criterion $\|r\|_2/\|\text{rhs}\|_2 < \epsilon$ taking the residual and the right-hand side into account is applied.

4 DISCUSSION OF POTENTIAL IMPLEMENTATION APPROACHES

Before describing the realized implementation in Section 5, this section discusses viable implementation options we considered. We first define design targets for the implementation in preCICE. Afterwards, we describe and discuss different programming libraries and frameworks that address our needs. Finally, we draw a conclusion on which the GPU-accelerated RBF interpolation in preCICE builds upon.

To realize GPU-accelerated data mapping in preCICE, we define the following design targets:

- The approach should take different execution backends into account. This means in particular the implementation should support vendor-independent execution on GPUs and multi-core CPUs.
- The maintenance effort of the solution should be as minimal as possible to guarantee a sustainable integration into preCICE.
- External dependencies should be established and available, e.g., through package managers, to ensure usability of the implementation.
- Since RBF interpolation involves linear-algebra, the dependency should offer corresponding data structures such as matrices and vectors as well as commonly used solvers and preconditioners through a high-level API.
- The implementation should be able to perform embarrassingly data-parallel operations such as the assembly of the coefficient matrix $\mathbf{C}$ with a custom kernel on the compute device to exploit the compute capabilities as well as possible.

We considered approaches to fulfill these design targets on various different levels. Since GPU programming is typically carried out through vendor-specific frameworks such as CUDA (provided by NVIDIA) or ROCm (provided by AMD), a straight-forward consideration would be a self-written implementation from scratch using these low-level frameworks. This approach introduces no external dependencies for preCICE, as all necessary ingredients are directly available on the associated hardware. However, these frameworks usually require an intricate memory-management and the implementation of sophisticated solver algorithms in an efficient way is a non-trivial task. In addition, any solution is bound to the hardware vendor, which leads to a substantial implementation effort to provide cross-platform performance portability. Instead of writing solver routines from scratch, a more obvious approach relies on functionality offered by hardware-vendor libraries such as cuSolver[2] or hipSolver[3]. These libraries expose a LAPACK-like interface to the user, which gives access to hardware-tailored highly-optimized solver routines. Although the libraries are separate software packages, they are shipped with the device libraries as well and introduce no visible dependency for preCICE.

To overcome vendor-specific implementation efforts, the OpenCL [10] standard provides a common abstraction layer for cross-platform parallel-programming of accelerator cards, e.g., GPUs or tensor processors. Low-level kernel code is compiled and deployed on the device at runtime, which ensures performance portability across different platforms. The OpenCL

[2]`https://docs.nvidia.com/cuda/cusolver/`

[3]`https://rocm.docs.amd.com/projects/hipSOLVER/`

standard is supported by almost all relevant hardware-providers and, similar to the device libraries, does not constitute a user-visible dependency. While OpenCL is very well-suited to parallelize algorithms in a data-parallel manner, the complexity for memory management and linear-algebra is comparable to an implementation using the device frameworks mentioned above.

SYCL [11] is another open standard that extends the core idea of OpenCL by expanding the supported computing platforms and reducing the API complexity. OpenCL is typically one backend used by SYCL itself to offer performance portability. The high-level interface mimics modern `C++` with focus on data-parallel algorithms such as a *parallel-for*. Although linear-algebra data structures and solvers are not directly available in SYCL, there are third-party software packages such as SYCL-BLAS, which implement matrix operations using SYCL itself. However, these packages usually introduce an external dependency in addition to SYCL.

Similar to SYCL, Kokkos [12] implements a `C++` interface for performance-portable algorithms and is well-established in the HPC community. Kokkos can be used, among other options, with SYCL operating as backend programming model. While the Kokkos core contains the abstraction layer for data structures, memory concepts as well as algorithms, the broader Kokkos ecosystem, e.g., the Kokkos kernels, directly support high-level math operations. Although Kokkos and the ecosystem packages can be installed on HPC systems via a package manager such as Spack, each component is a separate software dependency for preCICE.

Ginkgo [13] wraps the algorithmic complexity levels considered so far into a high-level linear algebra interface. Dedicated kernel implementations for CUDA (NVIDIA GPUs), HIP (AMD GPUs), DPCPP (Intel devices) and OpenMP ensure cross-platform performance with a focus on sparse linear systems, which are accessible through a common API. Conceptually, solvers and preconditioners are treated as operators in Ginkgo, which enables almost arbitrary configuration and combination options for the design of problem-tailored solver algorithms. Backend-specific kernels are statically compiled when building the library, but all compiled backends remain runtime configurable for the user. Similar to preCICE, Ginkgo is part of the Extreme-scale Scientific Software Development Kit [14] (xSDK), which ensures interoperability between these software components.

In summary, the potential implementation approaches for RBF interpolation in preCICE are vastly different and cover the complete range from vendor-specific low-level and highly-optimized implementations to ready-to-use cross-platform libraries operating on distinct complexity levels. We decided to follow and implement an RBF interpolation approach based on Ginkgo for various reasons: First and foremost, our focus lies on the application of linear-algebra to solve RBF systems and not the efficient implementation of linear-algebra subroutines, which is a non-trivial task itself. In this regard, Ginkgo is the only solution we considered offering a self-contained performance-portable ecosystem with a high-level abstraction layer for linear algebra. An additional benefit compared to other considered external dependencies is the superior interoperability between Ginkgo and preCICE as provided through xSDK and compliance with the xSDK software standards.

However, there are two shortcomings of Ginkgo compared to other approaches: First, the high-level interface does not provide any opportunity to dispatch and launch self-written matrix assembly kernels on the device (last design target). As a remedy, we employ an experimental non-released version of Ginkgo (branch *public_common_kernels*[4]) throughout this work, which exposes the internal kernel dispatch mechanism of Ginkgo to the API. Including this feature in a future release of Ginkgo is still in discussion. The second shortcoming of Ginkgo is the focus on sparse linear algebra. As explained in Section 5, we employ dense linear algebra in our implementation approach. Therefore, we complement the operators offered by Ginkgo with highly-optimized dense matrix solvers offered by cuSolver and hipSolver, supporting NVIDIA and AMD GPUs respectively. These solvers do not impose additional external dependencies for preCICE, as they are part of the vendor-specific software stack.

5 IMPLEMENTATION

This section deals with the implementation of the GPU-accelerated RBF interpolation in preCICE. Firstly, it starts with describing the computationally intense assembly of matrix $\mathbf{C}$ and optimization approaches to speed-up the kernel on the GPU. Secondly, it depicts the iterative solvers, including their preconditioners, that we implemented into preCICE to solve for the interpolation weights. Finally, as opposed to iterative solvers, the efficient application of direct solvers is discussed, namely a QR factorization that also leverages the compute power of GPUs. The current implementation targets multi-core CPUs or single GPUs and is not capable of combining the compute power of multi-GPUs in a mapping process. If preCICE is executed in an MPI-parallel environment, distributed data is handled using a gather-scatter approach, similar to the Eigen implementation.

5.1 RBF MATRIX ASSEMBLY

The assembly of the RBF coefficient matrix $\mathbf{C}$ is embarrassingly data-parallel because each matrix entry $\phi\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right)$ can be computed independently. The computational complexity of the assembly of $\mathbf{C}$ grows with $\mathcal{O}(N^2)$. As a result, a data-parallel kernel is developed by implementing and passing a lambda function into the kernel dispatch mechanism available in the experimental Ginkgo branch. During the build process of preCICE, the kernel is compiled for NVIDIA CUDA, AMD ROCm, and OpenMP backends. To exploit the massive amount of parallel executing threads accessible on GPUs, the dispatch mechanism creates $\mathcal{O}(N^2)$ threads; one for each matrix element. This is different to the OpenMP API, which spawns at most as many threads as CPU cores available. All available data points are split as equally as possible across the OpenMP threads.

There is still one issue that arises when writing general data-parallel code only once that should be executed on different platforms. That is, it is not optimized for efficiency on each individual executor. According to Cheng et al. [15], so-called *aligned coalesced memory accesses* are ideal when it comes to memory access patterns on GPUs. Whereas the term *aligned* refers to the first address of a memory transaction being a multiple of the cache granularity, *coalesced* denotes the fact that all threads within one warp access a contiguous chunk of memory. Especially the latter is of high interest in the case of matrix assembly because RBFs require the norm of the difference of two two- or three-dimensional vertices, i.e., every vertex is made up of two or three coordinates. When it comes to iterating over each dimension to calculate the norm in parallel, neighboring threads in the GPU thread grid should access neighboring memory addresses. Hence, each coordinate dimension is stored in a contiguous chunk of memory. On

[4]`https://github.com/ginkgo-project/ginkgo/commit/a195f856e`

top of that, the assembly kernel uses fused-multiply-add instructions to save one rounding step when performing a floating point multiplication that is immediately followed by an addition. This optimization is done for both CUDA and ROCm and realized via conditional compilation dependent on the currently compiled backend.

5.2 ITERATIVE SOLVERS

Ginkgo offers multiple linear iterative solvers. Few of them, that are widely used in scientific computing are the conjugate gradient (CG) method and the generalized minimal residual (GMRES) method. Both solvers can be configured to solve Eqn. (7). For the polynomial least-square system, a normal-equation is solved using a CG solver. All iterative solvers use a convergence criterion based on the residual norm reduction, which compares the norm of the initial residual, $\|r_0\|$, with the norm of the current residual, $\|r\|$, and stops if the relative reduction is below a predefined threshold, i.e., $\|r\|/\|r_0\| < \epsilon$. On top of that, in order to support CG and GRMES to approach a good solution faster, two out of several available preconditioners in Ginkgo are implemented into preCICE. Combining the solvers with preconditioners is beyond the scope of this work and we refer to [4] for more details in this context.

Furthermore, it is important to carefully select the best suited data structure for all matrices and vectors that are part of the RBF mapping. Ginkgo focuses on the iterative solution of sparse linear systems [13]. However, the sparseness of RBF system depends heavily on the support-radius of the applied basis function and non-uniform vertex clouds lead to irregular sparsity patterns, which render the efficient application of sparse data structures challenging. As a result, all the operations in preCICE are implemented using the dense matrix formats offered by Ginkgo.

5.3 DIRECT SOLVERS

The vendor-specific libraries cuSolver and hipSolver extend the facilities of Ginkgo to bring GPU-accelerated decompositions into preCICE. Both libraries offer different factorization methods such as QR and Cholesky decomposition in a LAPACK-derived naming scheme. As a starting point, a Householder QR decomposition is implemented and used for the coefficient matrix $\mathbf{C}$. Since Ginkgo data structures act as wrapper around memory spaces, they can easily be reused in the QR decomposition without the need for additional changes. Having computed the factorization once, preCICE can repeatedly compute the interpolation factors by using the matrix factors to solve a triangular system. Ginkgo offers a triangular solver out of the box that is used to compute the solution to this system every time a new right-hand side $\mathbf{f}$ is provided. For the polynomial system, however, an iterative CG solver is applied as described above.

6 RESULTS

We compare the new RBF interpolation approach with existing implementations using a turbine-blade geometry as a realistic example for coupled simulations. This section describes the numerical setup and shows performance results of the numerical experiments. All software and data components to reproduce the shown experiments are available in [16].

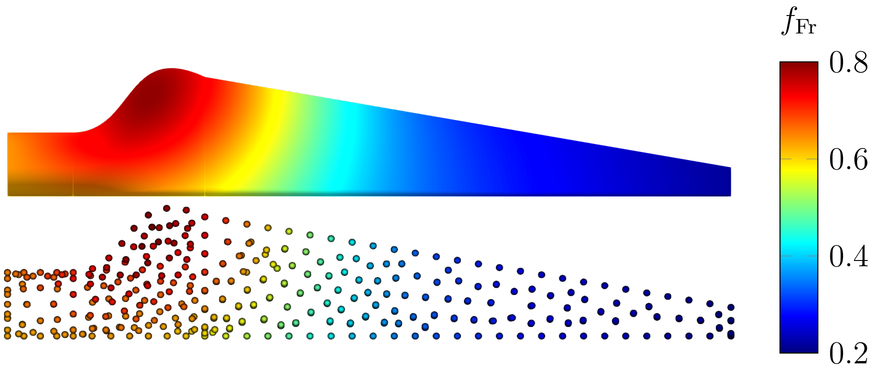

Figure 1: The turbine geometry and a corresponding mesh with data f_{Fr} stemming from Franke's function.

	Cuda CG	OpenMP CG	PETSc CG	Cuda QR	Eigen
backend	Ginkgo	Ginkgo	PETSc	Ginkgo/cuSolver	Eigen
preCICE exec.	Serial CPU	Serial CPU	64 CPU ranks	Serial CPU	Serial CPU
preCICE mapping	GPU	128 CPU threads	64 CPU ranks	GPU	Serial CPU

Table 1: Overview of compared backends and their execution environment. The mapping methods of Ginkgo and Eigen are always launched from a single CPU in preCICE, whereas the fully-parallelized PETSc backend is launched from a parallel preCICE. The CPU for all experiments is an AMD EPYC 7763 64-Core processor and the GPU is an Nvidia A100 GPU with 40 GB VRAM. We use PETSc version 3.18.5 and Eigen version 3.3.7.

6.1 SETUP

In order to investigate the implemented mapping approaches, we make use of the so-called Artificial Solver Testing Environment[5] (ASTE) provided by preCICE. ASTE is a thin wrapper around the preCICE API imitating two participants of a coupled simulation. Combined with the performance instrumentation of preCICE itself, ASTE enables insight into performance as well as accuracy metrics of different mapping configurations in preCICE.

Similar to [2], we use meshes of various refinement levels derived from a turbine-blade geometry [17] as shown in Figure 1. The mapping is executed for input meshes of different refinement levels, while keeping the output mesh at a constant size. The meshes were generated with Gmsh [18] by prescribing a target edge-length h between two adjacent vertices ranging from $h = 3 \cdot 10^{-2}$ (438 vertices) to $h = 4 \cdot 10^{-3}$ (21,283 vertices) for the input meshes and $h = 3 \cdot 10^{-3}$ (38,112 vertices) for the output mesh. Similar to [4], Franke's test function is sampled for each input mesh vertex in order to generate the relevant test data and serves as a reference solution for the interpolated values on the output mesh vertices.

The considered backends used for the mapping experiments are summarized in Table 1. Each backend is tested for two different stationary support radii: $\rho = 3h$ and $\rho = 20h$. Stationary support radius means that the radius is scaled along with the mesh width such that a fixed number of vertices is covered by the RBF support in radial direction, leading in our cases to an almost constant sparsity pattern within each configuration series. The iterative solver configurations (Cuda CG, OpenMP CG and PETSc CG) employ a conjugate-gradient solver (CG) with a convergence criterion $\epsilon = 1 \cdot 10^{-8}$. For iteratively solved polynomial contributions (cf. Eqn. (6))

[5]`https://github.com/precice/aste`

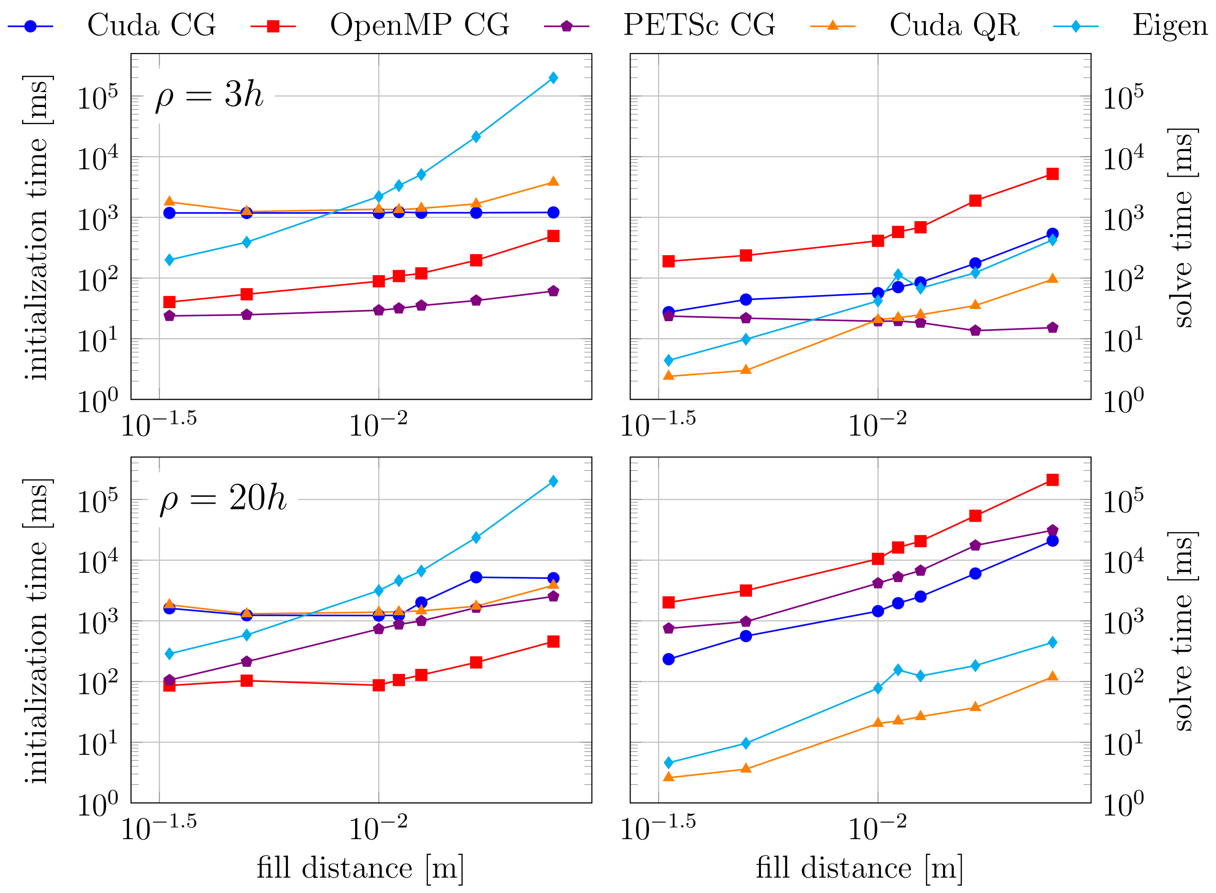

Figure 2: Initialization time (left column) and solve time (right column) with a support radius of $\rho = 3h$ (upper row) and $\rho = 20h$ (lower row). The initialization time is the one-time cost for a given vertex distribution and the solve time is the time required to interpolate values on the output mesh for given data on the input mesh. The shown timings are average timings obtained through five runs.

the CG employs a residual norm reduction of 10^{-4} as convergence criterion.

6.2 Performance Results

The performance results for both series are shown in Figure 2. For small problem sizes with less than 1,000 input mesh vertices, the CPU variants are faster than corresponding GPU computations, as the throughput-optimized GPU cannot operate at its full compute capabilities for such small problems. In fact, for small problem sizes the initialization time of the GPU dominates the overall initialization time of the mapping. However, apparent differences between the iterative methods relying on Ginkgo and PETSc might also stem from slightly different convergence criteria or implementations: While the Ginkgo-based solver requires around 200 to 400 iterations for $\rho = 3h$ and 3,800 to 9,700 iterations for $\rho = 20h$, the PETSc-based solver only requires around 70 to 150 and 3,800 to 6,000 iterations. The increased iteration count and associated runtime increase between $\rho = 3h$ and $\rho = 20h$ is due to an increase in the condition number by three to four orders of magnitude. In contrast, the runtime of direct solution methods is mostly defined through the size of the linear system and no observable performance changes can be noted between the two series when looking at the performance of Eigen or Cuda QR,

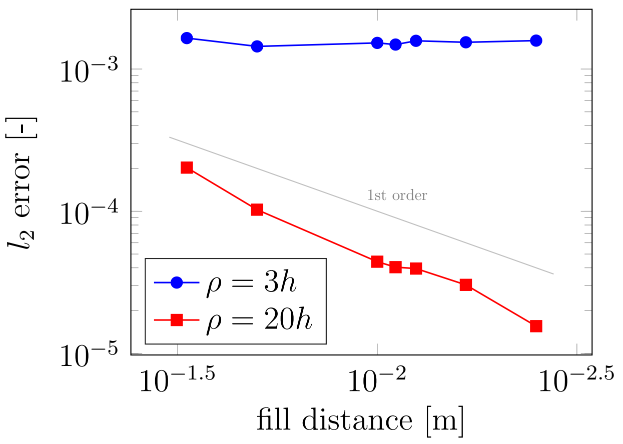

Figure 3: l_2 error $e = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\left(\mathcal{I}_f(\mathbf{x}_i) - f_{\mathrm{Fr}}(\mathbf{x}_i)\right)^2}$ associated to the timings shown in Figure 2. All runs produce the same error regardless of the executor. Note that stationary experiments eventually face error saturation and no convergence can be observed [5].

respectively. Therefore, iterative solvers are only competitive if the RBF configuration leads to a sufficiently well-conditioned linear system. However, as depicted in Figure 3, a smaller RBF support radius comes at the cost of a worse accuracy.

7 CONCLUSION

To conclude this paper, it is important to note that the GPU-based RBF interpolation fills a gap that existed in preCICE until now. In a very heterogeneous HPC landscape, there are many different setups ranging from many-core CPU environments to GPU compute clusters. Matching the available hardware with the resources required by individual simulations to reach a proper load-balancing in partitioned coupling is complex. In this respect, the presented approach provides additional flexibility by allowing the use of accelerator cards to solve computationally intense RBF problems in preCICE.

If the input and output meshes do not change throughout the simulation, direct solution strategies state an efficient and robust choice due to the large number of timesteps, which typically occur in coupled simulations. However, the algorithmic complexity of $\mathcal{O}(N^3)$ to compute dense matrix decompositions renders direct solution methods prohibitively expensive for large problem sizes. In such cases, iterative methods are oftentimes a more viable approach. From the user perspective, however, configuring RBF mappings with an iterative solver to find a good trade-off between accuracy and runtime can be a complicated procedure.

One drawback of all implementations introduced in this paper is that they act on dense matrix structures. When it comes to dealing with large problem sizes, the available VRAM might not be sufficient to store all required data structures and the approaches do not work any more. To overcome this issue, two approaches were tested in [4]. Firstly, there is *CUDA Unified Memory* that allows for overallocation by dynamically transferring memory from RAM to VRAM and vice versa as required and treating both as one large address space. Secondly, a matrix-free approach is implemented that does not store the coefficient matrix **C** explicitly but assembles it from scratch every time that it is needed by some algorithmic step, e.g., BLAS routines in CG. This builds upon the performance observations of the assembly kernel.

As a final remark, it is worth mentioning that RBF interpolation is only one mapping method available in preCICE. If the interpolation problem exceeds a certain size, globally-constructed RBF interpolation as discussed throughout this work is not appropriate any more, since the resulting linear system quickly becomes too ill-conditioned. Therefore, future work aims to combine the presented approaches with a partition-of-unity RBF method, which is already available as an MPI-parallel CPU implementation in preCICE. Enriching this available implementation with the support for accelerator cards discussed here promises to enable mapping computations on massively-parallel multi-node multi-GPU systems.

8 ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project number 515015468 and under Germany's Excellence Strategy - EXC 2075 – 390740016. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech).

REFERENCES

[1] H. Wendland, Scattered data approximation, Vol. 17, Cambridge University Press, 2004.

[2] G. Chourdakis, K. Davis, B. Rodenberg, M. Schulte, F. Simonis, B. Uekermann, G. Abrams, H.-J. Bungartz, L. Cheung Yau, I. Desai, K. Eder, R. Hertrich, F. Lindner, A. Rusch, D. Sashko, D. Schneider, A. Totounferoush, D. Volland, P. Vollmer, O. Z. Koseomur, preCICE v2: A sustainable and user-friendly coupling library [version 2; peer review: 2 approved], Open Research Europe 2 (51) (2022). `doi:10.12688/openreseurope.14445.2`.

[3] H.-J. Bungartz, F. Lindner, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, Partitioned fluid–structure–acoustics interaction on distributed data: Coupling via preCICE, in: H.-J. Bungartz, P. Neumann, W. E. Nagel (Eds.), Software for Exascale Computing - SPPEXA 2013-2015, Springer International Publishing, Cham, 2016, pp. 239–266. `doi:10.1007/978-3-319-40528-5_11`.

[4] T. P. Schrader, Efficient application of accelerator cards for the coupling library preCICE, Master's thesis, University of Stuttgart (2023). `doi:10.18419/opus-13027`.

[5] G. E. Fasshauer, Meshfree approximation methods with MATLAB, Vol. 6, World Scientific, 2007.

[6] F. Lindner, M. Mehl, B. Uekermann, Radial basis function interpolation for black-box multi-physics simulations, in: M. Papadrakakis, B. Schrefler, E. Onate (Eds.), VII International Conference on Computational Methods for Coupled Problems in Science and Engineering, 2017, pp. 1–12.
URL `http://hdl.handle.net/2117/190255`

[7] G. Guennebaud, B. Jacob, et al., Eigen v3, `http://eigen.tuxfamily.org` (2010).

[8] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. C. McInnes, R. T. Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp,

P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, J. Zhang, PETSc Web page, `https://petsc.org/` (2023).

[9] F. Lindner, Data transfer in partitioned multi-physics simulations: Interpolation & communication, Dissertation, University of Stuttgart (2019). `doi:10.18419/opus-10581`.

[10] J. E. Stone, D. Gohara, G. Shi, Opencl: A parallel programming standard for heterogeneous computing systems, Computing in Science & Engineering 12 (3) (2010) 66–73. `doi:10.1109/MCSE.2010.69`.

[11] The Khronos SYCL Working Group, Sycl 2020 specification (revision 4) (2021). URL `https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf`

[12] H. Carter Edwards, C. R. Trott, D. Sunderland, Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel and Distributed Computing 74 (12) (7 2014). `doi:10.1016/j.jpdc.2014.07.003`.

[13] H. Anzt, T. Cojean, G. Flegar, F. Göbel, T. Grützmacher, P. Nayak, T. Ribizel, Y. Tsai, E. S. Quintana-Ortí, Ginkgo: A Modern Linear Operator Algebra Framework for High Performance Computing, ACM Transactions on Mathematical Software 48 (1) (2022) 1–33. `doi:10.1145/3480935`.

[14] R. Bartlett, I. Demeshko, T. Gamblin, G. Hammond, M. A. Heroux, J. Johnson, A. Klinvex, X. Li, L. C. McInnes, J. D. Moulton, D. Osei-Kuffuor, J. Sarich, B. Smith, J. Willenbring, U. M. Yang, xSDK foundations: Toward an extreme-scale scientific software development kit, Supercomputing Frontiers and Innovations 4 (1) (2017) 69–82. `doi:10.14529/jsfi170104`.

[15] J. Cheng, M. Grossman, T. McKercher, Professional CUDA C Programming, EBL-Schweitzer, John Wiley & Sons, 2014.

[16] T. P. Schrader, D. Schneider, B. Uekermann, Replication Data for: Data-Parallel Radial-Basis Function Interpolation in preCICE (2023). `doi:10.18419/darus-3574`.

[17] F. Simonis, K. Davis, B. Uekermann, Test Setup of Turbine Blade Data Mapping (2022). `doi:10.18419/darus-2491`.

[18] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, International Journal for Numerical Methods in Engineering 79 (11) (2009) 1309–1331. `doi:10.1002/nme.2579`.