# Performance Optimization of Matrix-free Finite-Element Algorithms within deal.II

Martin Kronbichler
Technical University of Munich
Garching, Germany
kronbichler@lnm.mw.tum.de

Karl Ljungkvist
Uppsala University
Uppsala, Sweden

Momme Allalen
Leibniz Computing Centre
Garching, Germany

Martin Ohlerich
Leibniz Computing Centre
Garching, Germany

Igor Pasichnyk
IBM Germany
Garching, Germany

Wolfgang A. Wall
Technical University of Munich
Garching, Germany

## ABSTRACT

We present a performance comparison of highly tuned matrix-free finite element kernels from the `deal.II` finite element library on three contemporary computer architectures, an NVIDIA P100 GPU, an Intel Knights Landing Xeon Phi, and two multi-core Intel CPUs. The algorithms are based on fast integration on hexahedra using sum factorization techniques. On Cartesian meshes with a relatively high arithmetic intensity, the four architectures provide a surprisingly similar computational throughput. On curved meshes, the kernel is heavily memory bandwidth limited which reveals distinct differences between the architectures: the P100 is twice as fast as KNL, and almost four times as fast as the Haswell and Broadwell CPUs, effectively leveraging the higher memory bandwidth and the favorable shared memory programming model on the GPU.

## CCS CONCEPTS

• **Mathematics of computing** → Mathematical software performance;

**ACM Reference Format:**
Martin Kronbichler, Karl Ljungkvist, Momme Allalen, Martin Ohlerich, Igor Pasichnyk, and Wolfgang A. Wall. 2017. Performance Optimization of Matrix-free Finite-Element Algorithms within deal.II. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17).* ACM, New York, NY, USA, 3 pages.
https://doi.org/

## 1 INTRODUCTION

Matrix-free implementation of finite element operator evaluation, a technique originally established in the spectral element community, is the most competitive implementation for a wide range of applications relying on iterative solvers and quadratic or higher order shape functions [5, 7]. Computing the integrals on the fly trades a lower transfer of data from main memory for some additional arithmetic operations as compared to conventional solvers using sparse matrices. Since sparse matrix algebra is heavily memory bandwidth bound with arithemtic intensities of 0.16 to 0.25 floating point operations per byte loaded from main memory, a matrix-free alternative can be much faster also when performing more computations. A competitive evaluation of integrals is provided by sum factorization, a technique that decomposes the interpolation and unit cell derivatives for tensor product shape functions and quadrature into a series of 1D operations. This approach achieves arithmetic intensities between 1 and 10 floating point operations per byte loaded from main memory.

This work evaluates the performance of the general-purpose kernels included in the open-source finite element library `deal.II` [1] on contemporary hardware. It highlights the importance of memory bandwidth on performance, which gives the P100 GPU an edge over the competing architectures.


*SC'17, November 2017, Denver, Colorado USA*


## 2 MATRIX-FREE ALGORITHM

In both explicit time integration schemes and a selection of iterative solvers, the matrix-vector product is the primary determinant for application performance. This includes state-of-the-art multigrid solvers based on Jacobi and polynomial smoothers [7, 9].

Our matrix-free implementation considers a partial differential equation in weak form as the evaluation of the differential operator for a given input vector $u$ in terms of the underlying integrals. In the case of the Laplacian, we express the matrix-vector product $v = Au$ as

$$v_i = (Au)_i = \int_\Omega \nabla\varphi_i \cdot \nabla u^h \mathrm{d}x,$$

where the assembled system matrix $A$ is substituted by the underlying integrals with $\varphi_i$ the $i$-th test function and $u^h$ the piecewise polynomial representation associated with the input vector. In sum factorization, the matrix $A$ is neither built globally nor on the elements and only the integrals on the right hand side are evaluated by numerical quadrature in a loop over all cells in the mesh. The main arithmetic work is typically spent in the interpolation of the unit cell derivatives $\hat{\nabla} u^h$ given the vector entries $u_j$ on the cell and the multiplication and summation by $\hat{\nabla}\varphi_i$, see [5] and references therein. Sum factorization utilizes the tensor product structure in the shape functions and the quadrature formula by factoring out common factors along the respective directions and can thus apply a series of one-dimensional interpolation kernels along each coordinate direction, using ideas first presented in [10].

Sum factorization reduces the operational complexity from $O(k^d)$ operations per degree of freedom for $k$-th degree polynomials in $d$ dimensions for the naive evaluation or the final matrix stencil

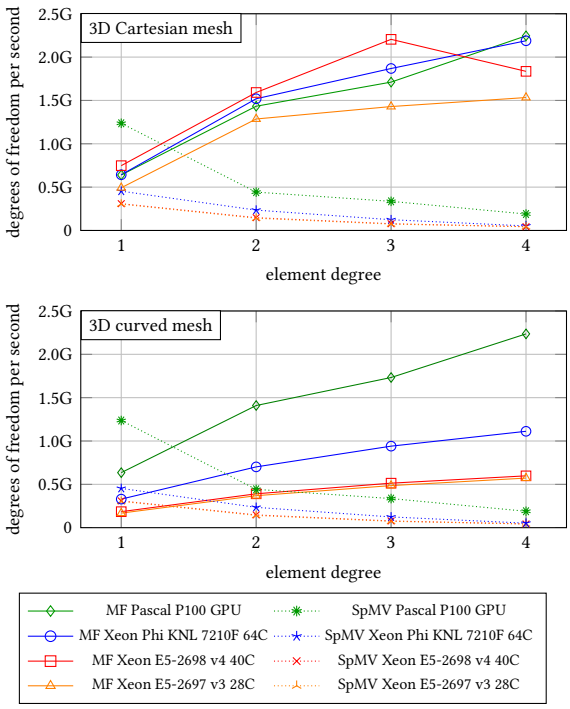


**Figure 1: Performance of operator evaluation on P100, KNL, and Xeon CPUs**

to $O(k)$ for all spatial dimensions. At degrees $k > 3$ in 3D, sum factorization hence does not only reduce the memory transfer, but also the number of arithmetic operations over matrices [5, 7].

Here, we evaluate the implementation of sum factorization within the `deal.II` finite element library [1] which has a particularly well-tuned CPU version that runs close to hardware limits [5–7]. The CPU version uses SIMD vectorization by C++ wrapper classes around intrinsics over several cells and gather/scatter operations for the vector access which were rewritten for AVX-512. Despite the high level of optimization, the kernels can be accessed in a generic way that can also be leveraged in complex application codes [3, 4]. For the GPU version, we extended the algorithms presented in [8, 9] with respect to shared memory parallelization and multiple GPUs. For the latter, we implemented explicit data exchange routines similar to the MPI implementation for the CPU [5]. Within a GPU, each elemental degree of freedom is assigned a separate thread in addition to the parallel for loop over elements, using atomics to avoid race conditions. This is opposed to the vectorization model over elements on the CPU and parallelization with MPI only. In our experiments, we found this to outperform thread-based implementations that schedule the integration tasks according to dependencies as a means to avoid race conditions [2].

## 3 PERFORMANCE COMPARISON

Fig. 1 displays the performance of the evaluation of the scalar Laplacian for degrees $1 \le k \le 4$ with $k+1$ quadrature points per space direction on a setup of one P100 (300W nominal power) within a DGX-1 system, a 64-core Intel Knights Landing Xeon Phi

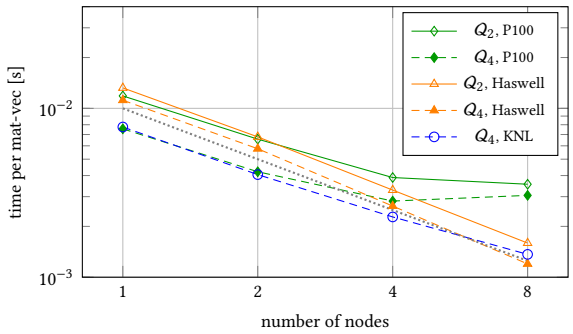


**Figure 2: Strong scaling experiment with 17.0 million degrees of freedom for Cartesian mesh on Xeon E5-2697 v3 nodes and P100 GPUs of DGX-1.**

7210F (230W nominal power) and $2 \times 20$ and $2 \times 14$ Broadwell and Haswell CPUs (270W nominal power each). For the measurements, we evaluated the operator on a discretization with around 10 million vector entries (exceeding caches) and display the data as the number of degrees of freedom processed per second. We record similar performance of the matrix-free kernels on all architectures when run on a Cartesian mesh with high arithmetic intensity around 3 to 8 [7]. In this case, only the input and output vector as well as some index data need to be fetched from main memory. When going to a mesh with high-order polynomial representation of a curved geometry, the P100 is almost twice as fast as KNL for $k \ge 2$ and almost 4× as fast as the CPUs. This is due to the much higher achieved memory throughput, which is measured at 375–430 GB/s on P100, 190–225 GB/s on KNL, and 95–125 GB/s on the Broadwell node with around 100 GFLOP/s on the CPUs and 400 GFLOP/s on P100. When compared to sparse-matrix kernels (SpMV), the matrix-free implementations offer speedups of an order of magnitude and more, showing the superiority of matrix-free techniques.

Fig. 2 displays a strong scaling experiment with 17 million degrees of freedom over multiple GPUs against the performance recorded on multiple nodes of the 28 core Haswell system. We see that the GPU implementation saturates when going from four to eight GPUs, which is due to missing parallelism and non-optimal data exchange routines. The CPU implementation with MPI only, on the other hand, does not only scale perfectly, but also sees a superlinear speedup when going from two to four nodes, which is due to a cache effect: for four and 8 nodes, the whole kernel fits into the L3 cache and can run somewhat faster.

## 4 CONCLUSIONS AND OUTLOOK

Our experiments show the great capabilities of the P100 GPU. While Intel CPUs and KNL can reach similar performance on compute-heavy Cartesian meshes, the P100 is able to considerably outperform the former in case memory bandwidth is important, which is a large share of matrix-free applications. Our results also show that the multi-GPU setup does not optimally use the available resources, shown by a breakdown in scaling below 3 milliseconds. Future work will investigate the latency limits of multi-GPU setups and implement better schemes for overlapping data transfer and computation, similar to the CPU implementation [5].

## REFERENCES


[1] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. 2017. The `deal.II` Library, Version 8.5. *J. Numer. Math.* 25, 3 (2017), 137–145. https://doi.org/10.1515/jnma-2017-0058

[2] Katharina Kormann and Martin Kronbichler. 2011. Parallel Finite Element Operator Application: Graph Partitioning and Coloring. In *Proceedings of the 7th IEEE International Conference on eScience*. 332–339.

[3] B. Krank, N. Fehn, W. A. Wall, and M. Kronbichler. 2017. A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow. *J. Comput. Phys.* in press (2017). https://doi.org/10.1016/j.jcp.2017.07.039

[4] M. Kronbichler, A. Diagne, and H. Holmgren. 2016. A fast massively parallel two-phase flow solver for the simulation of microfluidic chips. *Int. J. High Perf. Comput. Appl.* in press (2016). https://doi.org/10.1177/1094342016671790

[5] M. Kronbichler and K. Kormann. 2012. A generic interface for parallel cell-based finite element operator application. *Comput. Fluids* 63 (2012), 135–147. https://doi.org/10.1016/j.compfluid.2012.04.012

[6] M. Kronbichler, K. Kormann, I. Pasichnyk, and M. Allalen. 2017. Fast Matrix-Free Discontinuous Galerkin Kernels on Modern Computer Architectures. In *ISC High Performance 2017, LNCS 10266*, J. M. Kunkel, R. Yokota, and D. Balaji, P. Keyes (Eds.). Springer, Cham, 237–255. https://doi.org/10.1007/978-3-319-58667-013

[7] M. Kronbichler and W. A. Wall. 2016. A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. *arXiv preprint* arXiv:1611.03029 (2016).

[8] K. Ljungkvist. 2017. Matrix-Free Finite-Element Computations on Graphics Processors with Adaptively Refined Unstructured Meshes. In *HPC '17: Proceedings of the 25th High Performance Computing Symposium*. Society for Computer Simulation International, San Diego, CA, USA.

[9] K. Ljungkvist and M. Kronbichler. 2017. *Multigrid for Matrix-Free Finite Element Computations on Graphics Processors.* Technical Report 2017-006. Department of Information Technology, Uppsala University.

[10] S. A. Orszag. 1980. Spectral Methods for Problems in Complex Geometries. *J. Comput. Phys.* 37 (1980), 70–92.