# MaMiCo: Non-Local Means Filtering with Flexible Data-Flow for Coupling MD and CFD

Piet Jarmatz[0000−0002−5463−0740], Felix Maurer[0000−0003−4964−2103], and Philipp Neumann[0000−0001−8604−8846]

Chair for High Performance Computing, Helmut Schmidt University Hamburg, Germany, jarmatz@hsu-hh.de

**Abstract.** When a molecular dynamics (MD) simulation and a computational fluid dynamics (CFD) solver are coupled together to create a multiscale, molecular-continuum flow simulation, thermal noise fluctuations from the particle system can be a critical issue, so that noise filtering methods are required. Noise filters are one option to significantly reduce these fluctuations.

We present a modified variant of the Non-Local Means (NLM) algorithm for MD data. Originally developed for image processing, we extend NLM to a space-time formulation and discuss its implementation details.

The space-time NLM algorithm is incorporated into the Macro-Micro-Coupling tool (MaMiCo), a C++ molecular-continuum coupling framework, together with a novel flexible filtering subsystem. The latter can be used to configure and efficiently execute arbitrary data-flow chains of simulation data analytics modules or noise filters at runtime on an HPC system, even including python functions. We employ a coupling to a GPU-based Lattice Boltzmann solver running a vortex street scenario to show the benefits of our approach. Our results demonstrate that NLM has an excellent signal-to-noise ratio gain and is a superior method for extraction of macroscopic flow information from noisy fluctuating particle ensemble data.

**Keywords:** Flow Simulation · Non-Local Means · LBM · GPU · Software Design · Denoising · Data Analytics · Transient · Two-way Coupling · Molecular Dynamics · HPC · Molecular-Continuum

## 1 Introduction

Molecular dynamics (MD) simulations [16, 17] are used in many applications within computational science, engineering or biochemistry. They can be coupled to a computational fluid dynamics (CFD) solver, creating a multi-scale molecular-continuum flow simulation [2, 15]. On MD side, Brownian motion results in thermal fluctuations. Despite their importance and impact on molecular flow behavior, a direct incorporation of them into continuum-based simulations is not always desirable. It is rather the extraction of smooth flow data (including smoothed, i.e. averaged, Brownian effects) that might be sufficient here, i.e. to improve stability and performance of the coupled simulation [8]. One way to obtain this smooth data is ensemble averaging using many MD instances [13]. However, since MD is the computationally most challenging part in molecular-continuum setups, running many MD simulations is expensive. Various filtering methods including anisotropic median diffusion [12] and proper orthogonal decomposition (POD) [8, 9] have been proposed to handle molecular data. We have presented noise reduction using POD in 2019 [11] and introduced a noise reduction interface for the open source C++ coupling framework MaMiCo[1].

However, our latest research shows that POD filtering is rather sub-optimal in a molecular-continuum context: While POD is excellent for detecting temporal correlation in the input data, it does not fully exploit natural redundancy that exists in every single time step, because it views the MD data values as separate signal sources in the simulation space, without any information about their neighborhood relationships. Thus, in this paper we will propose a more advanced way to filter information from a particle system. Although the noise reduction interface we presented in our last paper [11] was already quite flexible, as it can be used to execute any noise filter or MD data analysis module, it could only run one filter at a time. However, it has been shown that in many cases a combination of more than one filter (e.g. POD + wavelet transform) results in improved results, see e.g. [21]. If many filters are to be combined, they may be interdependent and a free parametrization and reordering even at runtime is desirable. From an HPC perspective, this can be very problematic, as many challenges arise in terms of parallelization and communication or memory access (e.g. avoiding unnecessary copy operations). While any such filter combination can easily be hard-coded, it is desirable to strive for a more generic and flexible solution, with the ability to execute many noise filters and data analysis modules arranged in any data-flow diagram including multiple data paths, configurable at runtime, but also with a strong focus on performance and scalability on HPC clusters.

In this paper, we present an approach in which we view CFD cells and cell-wise MD data as 'pixels', execute various operations or chains of operations on them, including the application of an image processing denoising method: Non-Local Means (NLM) [4] first presented by Buades et al. [3] in 2004 is one of the best algorithms in this area. The ability to detect fine structures and patterns in images, sustaining them instead of averaging them away like many other filters would, makes NLM perfect for MD data, where one wants to separate nano-scale flow features from thermal noise. NLM has been successfully applied for medical imaging [5], but to the authors' knowledge it was never used in CFD/MD so far. In this paper, we present a modified version of the NLM algorithm, optimized for filtering particle data. Due to the non-local nature of the NLM method, performance can be a critical issue. Coupé et al. [5] have presented approaches for fast implementation of NLM, we optimize and extend them for higher-dimensional data and execution on HPC clusters. We also extend the C++ framework MaMiCo by new python bindings, our NLM implementation and we introduce a new filtering system for configurable filter combinations, creating flexible data-flow

[1] https://github.com/HSU-HPC/MaMiCo

filter chains, based on cell-wise operations. This system also enables the reuse of existing filters in MaMiCo, for instance from the python packages numpy and scipy, in an efficient and parallel way.

In Section 2 we explain our methodology for molecular-continuum coupling. Section 3 introduces the original Non-Local Means idea, some optimizations of it, and presents our new variant of the NLM algorithm. We also present a more flexible novel data-flow filtering subsystem for MaMiCo in Section 4, specifying details on software design and supported types of filters (4.1). Section 4.2 presents a fast implementation approach for NLM. We analyze NLM and the filtering subsystem more thoroughly by running a coupled molecular dynamics-Lattice Boltzmann vortex street flow test scenario on a GPU that is described in Section 5, giving simulation results, performance and signal-to-noise ratio measurements in Section 5.1. Section 6 summarizes our work, leading to perspectives for potential future research topics.

## 2 Coupled Simulation

We consider transient, concurrent coupling of MD and CFD, with nested time stepping on MD side (ca. 50 MD steps correspond to one CFD step).

On the molecular scale, our coupled flow simulation employs a simple Lennard-Jones particle system, integrating the equations of motion directly via explicit time stepping in the NVT ensemble. It is implemented using the linked cells method [7] and MaMiCo's built-in MD solver SimpleMD, which is used only to test the coupling algorithms and is representative for more sophisticated MD packages, such as LAMMPS [18] or ls1 mardyn [16]; see [14] for an overview of MD packages supported in MaMiCo-based coupling.

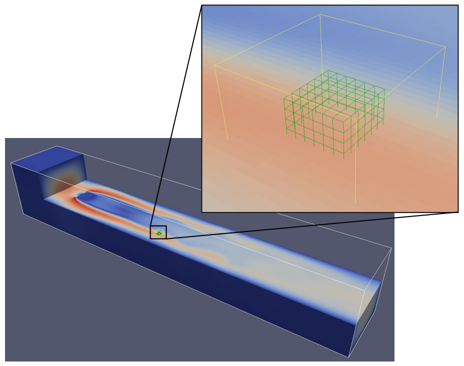
**Fig. 1.** Domain decomposition of coupled simulation. Zoom-out: 3D vortex street using 12.8 million Lattice Boltzmann cells. Zoom-in: Yellow outline: Total coupling region with ghost layer of macroscopic cells and outer MD cells. Green wireframe: Inner MD cells, where filtering is applied.

The MD domain is covered by a regular grid of cells which are used by MaMiCo for the coupling, called 'macroscopic cells'. Note that macroscopic cells neither have to match linked cells (they could even be used with a non-cell-based MD code) nor do they have to correspond to the cells used by the CFD solver internally. These cells are rather the main buffer data structure used for coupling and build the basis for all operations described in Section 4, where filters are always applied to multi-dimensional arrays of macroscopic cells.

On the continuum scale, similarly to the MD side MaMiCo can be coupled to many different solvers, but in the scope of this paper we use the lbmpy [1] package, which can automatically generate highly efficient GPU code for Lattice

Boltzmann (LB) schemes. We employ a simple single-relaxation-time collision operator and a D3Q19 stencil.

Figure 1 shows an exemplary domain decomposition, where a small MD domain is placed (overlapping) inside a much larger CFD domain to establish molecular-continuum coupling [13, 15]. The outer MD cell layers serve as a boundary condition to receive data from the continuum solver. We employ an advanced boundary forcing term [20] to model the thermodynamic pressure at the molecular scale at the continuum boundary and the USHER scheme [6] for particle insertion. In the inner MD cells, we sample cell-wise quantities such as density or average velocity out of the particle simulation instances, so that this data can be sent to the continuum solver. In the following, we will restrict considerations to these cells with regard to noise filtering.

## 3 Non-Local Means Algorithm

Buades et al. [3] proposed the Non-Local Means (NLM) algorithm after an extensive series of denoising experiments on natural images. It is inspired by various local smoothing methods and frequency domain filters, such as Gaussian convolution, anisotropic filtering, total variation minimization, Wiener filtering and wavelet thresholding. Already a very simple Gaussian filtering yields a powerful noise reduction by averaging away high-frequency noise, but it also removes high-frequency components of the original data; for instance sharp edges in an image are blurred. More advanced methods like an anisotropic filter can preserve large-scale geometrical structures like sharp edges, but all of these previously known methods still smooth out fine-scale textures. NLM on the contrary is constructed with the aim to preserve details and fine structures which behave like high-frequent noise in an image. This means that systematic patterns in particle data, for instance recurrent flow vortices or nano membrane diffusion structures will be retained by the filter instead of being averaged away.

For a noisy image $v = \{v(i) \mid i \in I\}$, NLM can be written as:

$$NLM[v](i) \;=\; \frac{1}{C_i} \sum_{j \in I} w(i,j) v(j) \tag{1}$$

We can see in Eq. (1) that the filtered result for the pixel at position $i$ is always a weighted sum of all other pixel values $v(j)$ in the image. The weights $w(i,j)$ are defined for all pairs of pixel positions $i$ and $j$; however, with the optimizations introduced in Sec. 4.2, $w(i,j)$ is sparse. The normalizing factor is chosen as

$$C_i \;=\; \sum_{j \in I} w(i,j) \tag{2}$$

so that the sum of weights is 1. Let $N_i$ denote a subset of $I$ that is a local neighborhood centered at position $i$. In [3] these neighborhoods in 2D images are often called *similarity windows* and have a size of, for instance, $7 \times 7$ pixels. On the other hand, [5] uses neighborhood windows in 3D images of size $3 \times 3 \times 3$

voxels. Here, we use the term *patch* for the small data array $v(N_i)$ in such a local neighborhood window $N_i$ around $i$. Our patches of MD data will normally have a size of $3 \times 3 \times 3 \times 3$ cells, corresponding to space-time windows, simply because a size of 3 is the smallest reasonable choice for both, space and time.

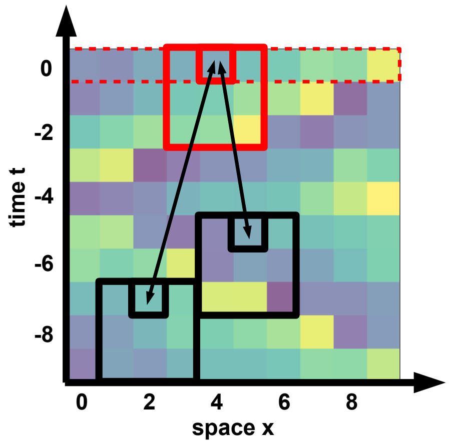

**Fig. 2.** NLM principle, illustrated for noisy 1D wave propagation: To compute the filtered value of a cell i at ($x$=4,$t$=0), its patch is compared with patches around every other cell j. Here, (2,-7) is more similar and thus gets a higher weight, while (5,-5) is less similar and has less influence on i.

In the temporal dimension, we restrict the filtering to a time window size of T coupling cycles (for example $T = 10$ in Fig. 2). We typically require filter results for the present cycle at $t = 0$, with data from past cycles $t < 0$ available, but before MD data for future time steps at $t > 0$ can be computed. Thus, the patches which are centered on their cell in space have to be shifted in time, as illustrated in Fig. 2. Additionally, we introduce a temporal fade-out of weights (see Eq. (5)), so that recent data is more significant than older information.

To compare patches, we use a squared Euclidean distance (L2 norm), as recommended by [4]. The norm can be understood as a similarity metric used to measure the similarity of i and j by comparing their local neighborhood patches:

$$\begin{aligned} (i,j) &= ((\boldsymbol{x}_i, 0), (\boldsymbol{x}_j, t)), t \in \{0, -1, -2, ..., -T+1\} && (3) \\ dist(i,j) &= |\hat{v}(N_i) - \hat{v}(N_j)|^2 && (4) \end{aligned}$$

This distance can be used to define the weights as:

$$w(i,j) = \exp\left(-\frac{max(dist(i,j) - 2\sigma_N^2, 0)}{h^2}\right) \cdot \frac{1}{1-t} \qquad (5)$$

Note that the expression $\exp(-\frac{d}{h^2})$ with $d \geq 0$ in Eq. (5) guarantees that all weights are in $[0,1]$ and that $h$ is a filtering parameter that controls the decay of the exponential function. The second filtering parameter $\sigma_N$ that corresponds to the expected standard deviation of the noise, so that the weight of a patch with a distance smaller than $2\sigma_N^2$ is always 1, as described in [4].

Unlike previously described NLM versions, in our variant we introduce an additional pre-filtering function $F$ (Eq. 6) into the similarity computation, inspired by regularization that is commonly used in a Perona-Malik filter. $F$ can be any existing simple noise filter with a low gain of signal-to-noise (SNR) ratio, such as a Gaussian kernel. It is used to improve the quality of the similarity metric in high noise / low SNR scenarios.

$$\hat{v} = F[v] \qquad (6)$$

Note that $F$ is not restricted to a Gaussian kernel, but we allow any reasonable pre-filter. This has the advantage that either a computationally cheap $F$ may be used or a filter with a higher SNR gain can be used, which improves the significance of the similarity metric. Especially, POD [8] can be applied here to exploit temporal correlation in $v$, and we show in Sec. 5.1 that this yields good results.

## 4 Implementation and Software Design

One of the most central concepts we developed with regard to creating a modular thus flexible filtering environment for MaMiCo is the notion of *filter sequences*. Let $\mathbb{F}$ be a set of filters operating on the same spatial domain. Then one can view a filter sequence $S$ as a nonempty tuple or list $(f_0, .., f_n : f_i \in \mathbb{F})$ in which for all $i \geq 1$ the input of $f_i$ shall be the output of $f_{i-1}$. The output of $f_n$ is the sequence output of $S$. Similarly, the input of $f_0$ is the sequence input of $S$. The sequence input of all sequences must be defined before execution, it can be either MD data or another sequence.

In some cases, being restricted to unary functions as filters poses a problem. For example, our NLM algorithm asks for two sets of input (unfiltered/prefiltered). We thus implemented a generalization of the sequence concept, allowing for multiple in- and outputs: *filter junctions*. Filters that operate not in sequences but in junctions are called *junctors*. An exemplary filtering configuration using both junctors and filters is depicted in Fig. 4.

Sequences are managed by `FilteringService`, which also acts as an interface to the `MacroscopicCellService` (MCS). During MaMiCo startup, one MCS per MD instance gets initialized. It immediately constructs an instance of `FilteringService`. This service now interprets an XML configuration file, creating all `FilterSequences` specified. Every sequence is now instructed which cells will be the input for $f_0$ and where to write the output of $f_n$. Then, filter instantiation begins. Internally, each sequence stores two cell vectors $V_1$ and $V_2$. Let $I(f_i)$ denote the input and $O(f_i)$ the output of a filter $f_i$ respectively. Then,

$$I(f_i) = \begin{cases} V_1 & \text{if } i \text{ is even} \\ V_2 & \text{if } i \text{ is odd} \end{cases} ; \quad O(f_i) = \begin{cases} V_2 & \text{if } i \text{ is even} \\ V_1 & \text{if } i \text{ is odd} \end{cases} .$$

This way, space required for cell vectors is constant in $n$.

### 4.1 Supported Types of Filters

For a $D$-dimensional scenario, any function applicable to both scalar and $D$-dimensional floating point in- and outputs can be utilized as a filter or junctor. This also means that read-only utilities can easily be wrapped in filters. For example, saving cell data to files is implemented as a `WriteToFile` filter, cf. Fig. 3. Such filters simply copy input cell data to corresponding output cells without prior modification.

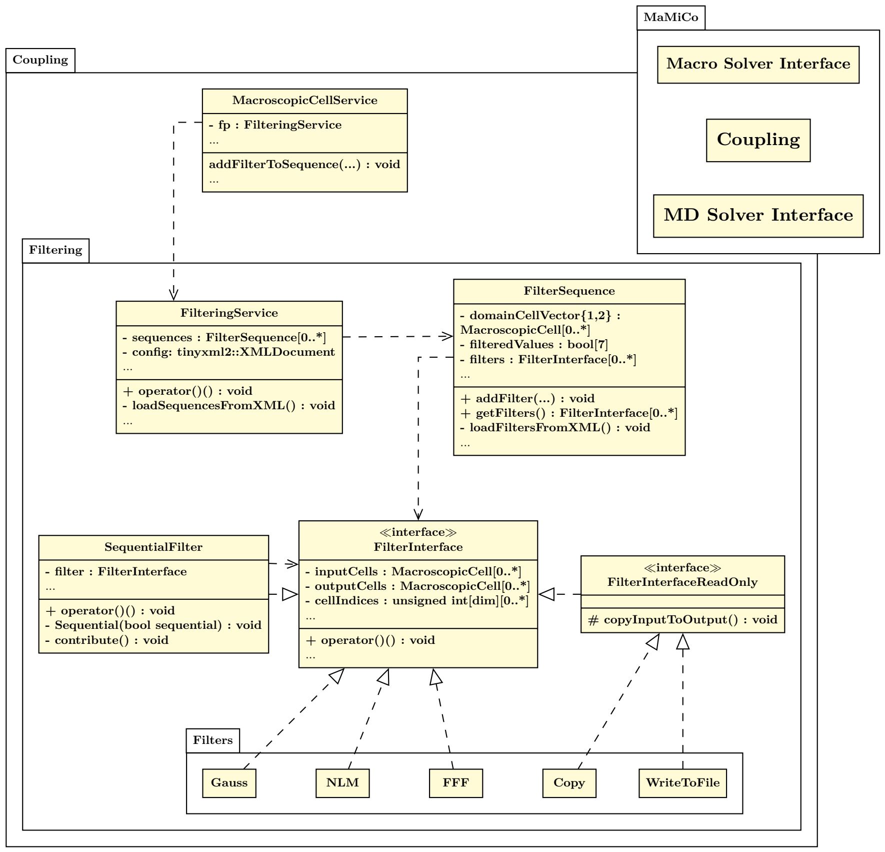
MaMiCo
Macro Solver Interface
Coupling
MD Solver Interface
Coupling
MacroscopicCellService
- fp : FilteringService
...
addFilterToSequence(...) : void
...
Filtering
FilteringService
- sequences : FilterSequence[0..*]
- config: tinyxml2::XMLDocument
...
+ operator()() : void
- loadSequencesFromXML() : void
...
FilterSequence
- domainCellVector{1,2} : MacroscopicCell[0..*]
- filteredValues : bool[?]
- filters : FilterInterface[0..*]
...
+ addFilter(...) : void
+ getFilters() : FilterInterface[0..*]
- loadFiltersFromXML() : void
...
SequentialFilter
- filter : FilterInterface
...
+ operator()() : void
- Sequential(bool sequential) : void
- contribute() : void
≪interface≫
FilterInterface
- inputCells : MacroscopicCell[0..*]
- outputCells : MacroscopicCell[0..*]
- cellIndices : unsigned int[dim][0..*]
...
+ operator()() : void
...
≪interface≫
FilterInterfaceReadOnly
# copyInputToOutput() : void
Filters
Gauss
NLM
FFF
Copy
WriteToFile

**Fig. 3.** Software design of the new filtering system for MaMiCo: Every filter implements the FilterInterface, they are combined together in FilterSequences and managed by the FilteringService.

**Statically and Dynamically Linked Filters** As mentioned before, the main way of defining filters in a `FilterSequence` is via a configuration file. Naturally, the tree-like node structure of XML is very close to the structure of interlinked `FilterSequences`, cf. 4. Filters specified before runtime using this configuration file are called *statically linked*. These filters are written in C++, implementing the `FilterInterface` or expanded interfaces thereof. *Dynamically linked* filters on the other hand are linked at runtime by calling the method `addFilter()` of a `FilterSequence`. This method takes arbitrary `std::function` objects (of signatures fitting our definition of filters above) and constructs a `FilterFromFunction` (FFF) C++ object using that information. Since there are interfaces between `std::function` and other programming languages, this design omits the limitation of filters to be written in C++. For example, the pybind11 library [10]

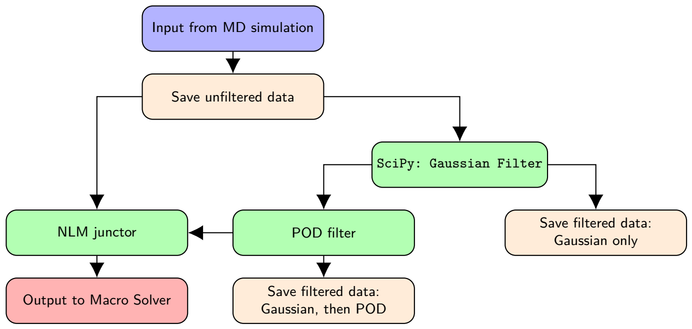

**Fig. 4.** Visualization of an exemplary *FilteringService* configuration diagram. Setups like this enable comparison between any number of filtering strategies with ease, e.g. between Non-Local Means, Gaussian and Gaussian followed by POD. Note that NLM is implemented as a junctor: it uses both, unfiltered data and POD output.

allows for python functions to be passed as `std::function` at runtime. Listing 1 gives an example as to how one could make use of SciPy[2] filters in MaMiCo.

```python
from scipy.ndimage import gaussian_filter

def gaussSigmaOne(data):
    print("Applying gaussian filter. sigma_G = 1.")
    return gaussian_filter(data, sigma = (1,1,1))

mcs = self.multiMDCellService.getMacroscopicCellService(0)

#Add scipy's gaussian filter in FilterSequence 'mySequence'
#at filter index 0, filtering scalar properties.
mcs.addFilterToSequence(filter_sequence = "mySequence", filter_index =
↪  0, scalar_filter_func = gaussSigmaOne, vector_filter_func = None)
```

Listing 1: Example showing how to use the MaMiCo python interface in order to add an arbitrary function as a filter at runtime. In this case, the `SciPy` package is utilized to apply a Gaussian filter to all scalar properties "mySequence" (defined via XML) operates on. With "`filter_index=0`" the filter is placed at $f_0$. `mcs` is the overarching `MacroscopicCellService`, cf. Fig. 3

**Sequentialized Filters in a Parallel Environment** One of MaMiCo's primary design goals is to support massively parallel solvers and thus be highly scalable. In particular, the MD simulation will usually be MPI-parallel. In that case, each MPI rank will have a unique instance of MCS and thus of `FilteringService`, managing filters entirely independent of other ranks. These filters will then only

[2] https://www.scipy.org/

have a fraction of all cells in the global filtering domain, which can, depending on individual use cases, have either a negative or positive effect:

Some filter algorithms treat cells independently, in which case parallel filtering can imply performance gain. In other cases this separation is either suboptimal or entirely incompatible with the filter algorithm in use.

For the latter case we introduced the concept of *sequentialized filters*. If a filter is marked as *sequential*, that filter will not be applied in rank-wise independence. Instead, it has only one dedicated *processing rank*, while all other ranks are *contributors*. For application of a sequentialized filter, the `FilteringService` uses `MPI_Gather` and `MPI_Bcast` to manage the necessary communication steps. Although sequentialized filters represent only a special case within our methodology, they allow for simple incorporation of off-the-shelf filter implementations. E.g. a dynamically linked Gaussian filter can be used without having to manually manage MPI communication for neighbor information.

## 4.2 Non-Local Means: Optimized Implementation

Despite the name 'non-local', NLM is in practice not executed on the global domain, but the similarity comparison of patches is restricted to moving search windows. This means that $j$ in Eq. (1) is not actually in $I$, but $\boldsymbol{x}_j$ is in a local neighborhood of constant spatial size $(2M+1)^3$, centered at $i$. Thus, since only a constant number of operations has to be performed in this local research zone, the computational complexity of the NLM implementation is asymptotically linear in the global number of domain cells.

To speed up the implementation, as described in [5], the computation of $w(i,j)$ can be skipped if $dist(i,j)$ is expected to be large, without the expensive evaluation of $|\hat{v}(N_i) - \hat{v}(N_j)|^2$. This is achieved by pre-computing and storing characteristic values of $\hat{v}(N_i)$ for every cell $i$, such as mean and standard deviation of this patch. Only if the relative differences of these values are small enough, i.e. inside configured bounds, then $dist(i,j)$ is computed, otherwise $w(i,j)$ is set to zero.

At the expense of extra memory consumption, the NLM execution can be further accelerated if patches are stored in a redundant, linearized way: This allows for a cache-efficient contiguous memory access and vectorization of the patch distance determination. Our implementation exploits this in a natural way – using the concept of *flowfield* and *patchfield* data structures. We define a *field* as a four-dimensional space-time array. Then, a flowfield is a field of quantities sampled from the particle system. A *patch* is a small flowfield together with characteristic values; and a patchfield is a field of patches. Our NLM implementation only constructs and accesses a patchfield, so that for the similarity computation, it does not have to use a cell stencil or compute any neighbor indices. In our experiments we measured a speed-up of 20 % compared to direct neighbor cell accesses without linearized patch storage.

Since the filtering system of MaMiCo executes the filter only on the subdomain of inner macroscopic cells that belongs to this rank, the NLM implementation is MPI-parallel and can easily be scaled up to runs on large HPC-clusters.

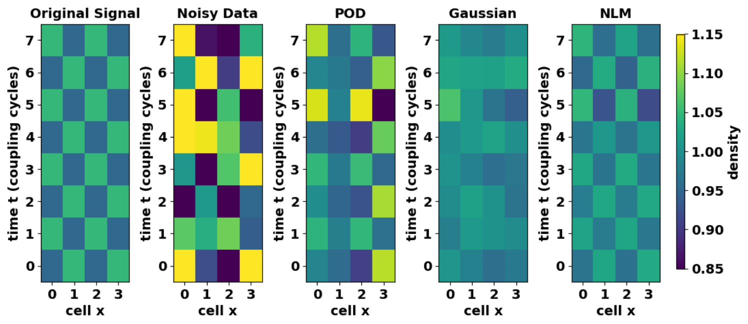

**Fig. 5.** Sound wave extraction test case: even with a very high level of noise in the input, our NLM implementation (using POD as pre-filter) can reconstruct the original pattern in the density data. NLM params used: $\sigma_N = 0$, $h^2 = 0.45$

However, since we did not incorporate ghost layers for the patchfield initialization so far, the filtering quality at inter-rank boundaries can be slightly reduced due to missing neighbor values. For simplicity, we test NLM sequentially here.

### 4.3 NLM Test: Nano-Pattern (Sound Wave) Extraction

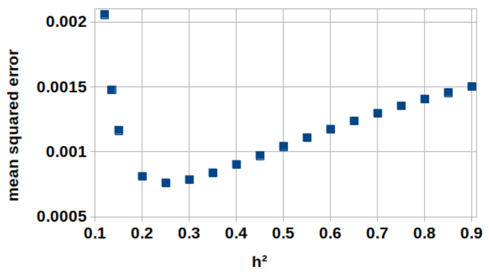

**Fig. 6.** NLM parametrization study: MSE against original signal (Fig. 5) over $h^2$

To test the filters under challenging conditions and demonstrate the ability of NLM to detect fine structures in MD data, we created a 3D flow scenario where ultra-high-frequent pressure wavefronts propagate at speed of sound (one cell per coupling cycle) in X direction through the domain. For simplicity, and to have an original true signal for comparison, we did not execute a real particle simulation here, but constructed synthetic pseudo-MD data by adding normally distributed artificial noise ($\sigma = 0.2$) to analytical oscillations (amplitude 0.05). Figure 5 shows the density values in four inner cells of the synthetic MD domain over time, and compares the noisy input with the respective filter outputs. Since we normalized the wavelength to two cell sizes, the original signal shows a checkerboard pattern in the $x - t$ plot. The filtering parameters $N$,$k_{max}$,$\sigma_G$,$T$ and $F$ are exactly the same as in Sec. 5.1. While it can be observed that the Gaussian filter and POD [11] deliver blurry results, the new NLM approach captures the spatially varying pattern very well. Figure 6 shows the same test setup again, but evaluates the mean squared error (MSE) of the NLM output for various choices for the filtering parameter $h^2$. Each MSE value is computed over 64 cells and 50 coupling cycles here. We can identify an optimal value of $h^2 \approx 0.25$ for this scenario, and discover that too large values of $h$ are less critical, while with too small values the filtering results fall off in quality much faster.

## 5 Simulation Results

**Vortex Street Test Scenario** We chose a well-known vortex street setup for testing the filtering system and the NLM implementation in a transient one-way LB $\rightarrow$ MD coupled simulation, i.e. without sending data from MD to the macroscopic solver, so that we obtain reproducible results for validation and quantification of the denoising.

We use lbmpy [1] as macroscopic solver to set up the test case *3D-2Q* from [19] that models a flow around a cylinder with square cross-section, but we scale it down to a channel of size $663\,\text{nm} \times 108\,\text{nm} \times 108\,\text{nm}$. We keep the Reynolds number of $Re = 100$ and thus the flow properties constant, so that the flow is laminar and unsteady. More details defining the scenario precisely can be found in [19]. To validate the flow scenario, we measured the forces acting on the obstacle to compute drag and lift coefficients. Our experimentally determined values of $c_{D_{max}} = 4.39; c_{L_{max}} = 0.066$, as well as the Strouhal number we measured of $St = 0.37 \pm 0.03$ are in very good compliance with the results given in [19], which were obtained by multiple different research groups and using other numerical methods, such as finite element solvers. The LB simulation uses $780 \times 127 \times 127$ cells and runs with a performance of 1937 MLUPS on a NVIDIA Tesla V100.

Using the novel MaMiCo python bindings, we couple lbmpy to a single MD instance placed at the position $332\,\text{nm} \times 80\,\text{nm} \times 54\,\text{nm}$ in the vortex street, as shown in Fig. 1. We ensure that the dynamic viscosity is configured consistently between both solvers and for simplicity, we choose the same cell size for both MD and LB. Note that while the vortex street is not a common scenario in nanofluidics, it yields a multidimensional transient flow in the coupled simulation, which is excellent for studying the filtering methods on sufficiently complex particle data.

### 5.1 Vortex Street Filtering Results

We initialize the scenario described in Sec. 5 for only $2 \times 10^4$ LB time steps, so that there is no fully developed flow, but a start-up phase can be observed. At that point we enable coupling, sample and filter particle data for 1000 coupling cycles (with 100 MD time steps per cycle). The resulting values for the velocity in Y direction in one of the cells in the center of the inner MD domain are shown in Fig. 7. The main flow in X direction is not shown here, but instead only unsteady oscillating transversal fluid movements behind the obstacle (cf. Fig. 1).

It can be seen in Fig. 7 that without filtering, there is a high level of thermal noise in the values sampled from MD. With all filtering methods, the results are much closer to the CFD values. We can quantify this precisely with the signal-to-noise ratio (as defined in [11]) in this cell: While the raw MD data has

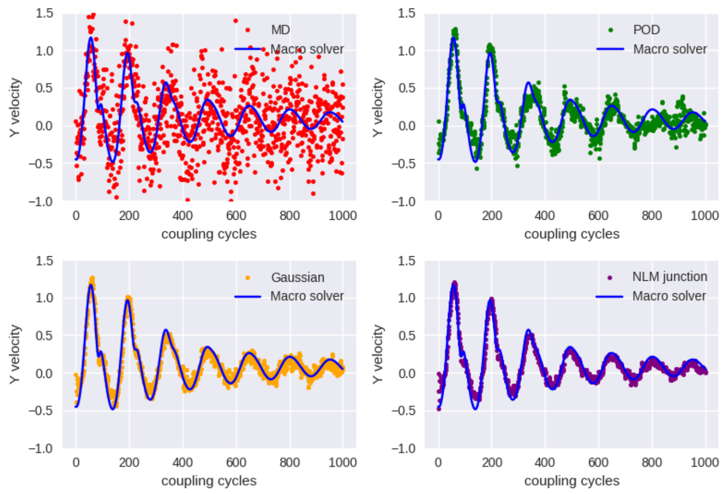

**Fig. 7.** Velocity data from particle system, compared to macroscopic flow, in only one single cell (center of MD domain). Red: raw MD (1 instance). Green: POD filtered, $N = 40$, $k_{max} = 2$. Yellow: Gaussian, $\sigma_G = (1, 1, 1)$. Purple: NLM, $T = 5$, $F$=POD.

an SNR of -3.16 dB, this is improved to 6.06 dB with POD, 9.62 dB with the Gaussian filter and 10.57 dB with NLM. NLM is configured to use POD as pre-filter $F$. Compared to Sec. 4.3, the Gaussian performs better here, because in this example the flow has rather a large-scale structure instead of fine patterns, but NLM is still superior. Note that the time window size used is 40 coupling cycles for POD but only 5 for NLM, so that POD has 8 times more data available to use for filtering. Nevertheless, NLM achieves an increased SNR gain of 4.51 dB, corresponding to 2.82 times stronger noise reduction compared to POD.

The performance impact of the filtering system on the overall coupled simulation is minimal. For the configuration in Fig. 7 we measured the following average filter execution times per cycle, for all cells: POD 1.03 ms, Gaussian 0.33 ms, NLM 6.98 ms. This corresponds to less than 0.2 % of total simulation runtime spent in the filtering system in the worst case. For more details on performance and scaling tests on up to 512 cores of the coupled simulation including POD the reader is referred to [11].

## 6 Conclusions

We have discovered that the NLM algorithm from image processing is by design well suited to filter flow data from a MD system. When we generalized NLM to higher-dimensional and time series data, the problem of missing information from the future was solved by introduction of a time-shift (Fig. 2) in the definition of patches. We allow an additional pre-filtering step for an improved similarity metric in high noise setups and provided a formal definition of our new NLM variant. Although it operates on 4D data structures, it is efficient, especially compared to multi-instance MD computations, but some questions such as more elegant boundary handling and good parallelization strategies still have to be investigated. To be able to execute many data analysis or noise filtering modules in freely configurable combinations, we introduced a more flexible novel filtering system into MaMiCo and gave details on the new software design. We have shown how NLM can be efficiently implemented and provide an optimized implementation as junctor in MaMiCo. To allow for dynamically linked filters and reuse of existing python packages in the C++ framework, we extended MaMiCo by new python bindings. We introduced a coupling to lbmpy, validated a GPU-based vortex street test scenario, and used the vortex street flow to investigate NLM results and quantify the noise reduction, showing that NLM is superior to POD in a molecular-continuum context, at minimal performance expenses.

A prediction of the deviations between microscopic and continuum quantities is important for consistency in the coupling, i.e. to guarantee conservation of energy. This can easily be derived from statistical mechanics for raw MD data and ensemble averaging, but for filtering results such a generic error estimation tool constitutes a major challenge. Machine learning methods may be an approach to solve this. Other open questions for future work would be further investigation of possible limitations and drawbacks of the pattern extraction by NLM, as well as automatic dynamical selection of optimal filtering parameters to obtain a balanced ratio between and performance and accuracy.

**Acknowledgments** Parts of this work have been supported through the HSU-IFF project "Resilience and Dynamic Noise Reduction at Exascale for Multiscale Simulation Coupling" and the dtec.bw project "MaST". We also want to thank the lbmpy developer Martin Bauer for fruitful discussions and support at using the software.

## Bibliography

[1] Bauer, M., Köstler, H., Rüde, U.: lbmpy: Automatic code generation for efficient parallel lattice boltzmann methods. Journal of Computational Science **49**, 101269 (2021)
[2] Borg, M.K., Lockerby, D.A., Reese, J.M.: A hybrid molecular–continuum method for unsteady compressible multiscale flows. Journal of Fluid Mechanics **768**, 388–414 (2015)
[3] Buades, A., Coll, B., Morel, J.M.: On image denoising methods. CMLA Preprint **5** (2004)
[4] Buades, A., Coll, B., Morel, J.M.: Non-local means denoising. Image Processing On Line **1**, 208–212 (2011)
[5] Coupé, P., Yger, P., Barillot, C.: Fast non local means denoising for 3d mr images. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 33–40, Springer (2006)

[6] Delgado-Buscalioni, R., Coveney, P.: USHER: an algorithm for particle insertion in dense fluids. The Journal of chemical physics **119**(2) (2003)

[7] Gonnet, P.: A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations. Journal of Computational Chemistry **28**(2), 570–573 (2007)

[8] Grinberg, L.: Proper orthogonal decomposition of atomistic flow simulations. Journal of Computational Physics **231**(16), 5542–5556 (2012)

[9] Grinberg, L., Yakhot, A., Karniadakis, G.E.: Analyzing transient turbulence in a stenosed carotid artery by proper orthogonal decomposition. Annals of biomedical engineering **37**(11), 2200–2217 (2009)

[10] Jakob, W., Rhinelander, J., Moldovan, D.: pybind11–seamless operability between c++ 11 and python. URL: https://github. com/pybind/pybind11 (2017)

[11] Jarmatz, P., Neumann, P.: Mamico: Parallel noise reduction for multi-instance molecular-continuum flow simulation. In: International Conference on Computational Science, pp. 451–464, Springer (2019)

[12] Ling, H., Bovik, A.C.: Smoothing low-snr molecular images via anisotropic median-diffusion. IEEE transactions on medical imaging **21**(4), 377–384 (2002)

[13] Neumann, P., Bian, X.: Mamico: Transient multi-instance molecular-continuum flow simulation on supercomputers. Computer Physics Communications **220**, 390–402 (2017)

[14] Neumann, P., Flohr, H., Arora, R., Jarmatz, P., Tchipev, N., Bungartz, H.J.: Mamico: Software design for parallel molecular-continuum flow simulations. Computer Physics Communications **200**, 324–335 (2016), ISSN 0010-4655

[15] Nie, X., Chen, S., Robbins, M.O., et al.: A continuum and molecular dynamics hybrid method for micro-and nano-fluid flow. Journal of Fluid Mechanics **500**, 55–64 (2004)

[16] Niethammer, C., Becker, S., Bernreuther, M., Buchholz, M., Eckhardt, W., Heinecke, A., Werth, S., Bungartz, H.J., Glass, C.W., Hasse, H., et al.: ls1 mardyn: The massively parallel molecular dynamics code for large systems. Journal of chemical theory and computation **10**(10), 4455–4464 (2014)

[17] Páll, S., Abraham, M.J., Kutzner, C., Hess, B., Lindahl, E.: Tackling exascale software challenges in molecular dynamics simulations with gromacs. In: EASC, pp. 3–27, Springer (2014)

[18] Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. Journal of Computational Physics **117**(1), 1–19 (1995), ISSN 0021-9991

[19] Schäfer, M., Turek, S., Durst, F., Krause, E., Rannacher, R.: Benchmark computations of laminar flow around a cylinder. In: Flow simulation with high-performance computers II, pp. 547–566, Springer (1996)

[20] Zhou, W., Luan, H., He, Y., Sun, J., Tao, W.: A study on boundary force model used in multiscale simulations with non-periodic boundary condition. Microfluidics and nanofluidics **16**(3), 1 (2014)

[21] Zimoń, M., Prosser, R., Emerson, D., Borg, M.K., Bray, D., Grinberg, L., Reese, J.M.: An evaluation of noise reduction algorithms for particle-based fluid simulations in multi-scale applications. Journal of Computational Physics **325**, 380–394 (2016)