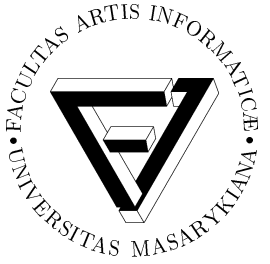
Software Performance Optimization in Scientific Computing

Jiří Filipovič



Habilitation Thesis

2023

Acknowledgments

I would like to express my deep thanks to my advisors Prof. Luděk Matyska and Prof. Siegfried Benkner, for their guidance and comments on my work during my postdoc at Masaryk University and the University of Vienna. I would also like to thank my interdisciplinary collaborators: the group of Prof. Jiří Damborský for collaboration on the simulation of transport processes in proteins, and the group of Prof. José M. Carazo and Dr. Carlos S. Sorzano for collaboration on GPU acceleration of image processing in cryo-electron microscopy. My deep thanks also belong to my current and former colleagues, who I had the pleasure to work and publish with: Dr. Jana Hozzová, Dr. David Střelák, Filip Petrovič, Jaroslav Oĺha, Petra Němcová, Dr. Amin Nezarat, Dr. Jan Fousek, Jan Plhák, David Myška, Jan Polák, and Jakub Kiska.

Last but not least, my special thanks belong to my family for being patient with me when I was working on my research.

Abstract

Scientific computing is considered to be the third mode of science, complementing experiments and theory. It uses simulation and modeling to understand complex problems, make predictions, or analyze hypothetical situations. In many practical settings, the computational experiment requires processing complex models, introducing a vast amount of computations, and, hence, high demands for hardware resources. Therefore, multiple approaches are applied to make those models computationally feasible. Besides searching for algorithms in a better time complexity class, results can be approximated by employing computationally less demanding models, or high-performance computing methods can be used to optimize and scale algorithms on parallel computers.

This thesis surveys the author's contribution to multiple methods for improving the computational efficiency of scientific computing applications. We focus on optimizing algorithms where their alternatives in better time complexity are unknown. We apply a holistic top-down approach, decreasing the computational cost by leveraging mathematically different and computationally less demanding models and bringing speedup by source code optimization and acceleration on GPUs. We generalize the results when possible, so they are applicable in different problem domains.

Employing different approximative models is described in two use-cases from computational biology: *analysis of transport processes in proteins* and *haptic interactions with deformable models of tissues*. In the first case, we introduce a novel path-finding technique for searching many-dimensional spaces of molecular movement and conformational changes as an alternative to more computationally expensive molecular dynamics. In the second case, we employ a precomputation-interpolation scheme as an alternative for real-time deformation computation, so a model that is too complex for real-time updating can be used in interactive simulations.

GPU acceleration is examined in two problem domains: *GPU-accelerated molecular modeling* and *GPU-accelerated image reconstruction in cryo-electron microscopy*. The GPU accelerated implementation typically requires not only parallelization of the accelerated algorithm but also its re-formulation in order to fit the GPU architecture better. In contrast to altering a computational model, many GPU-accelerated codes share optimization strategies. Therefore, we generalize certain code optimization tasks typically applied in GPU acceleration and describe our contribution to the state-of-the-art in *kernel fusion* (code transformation improving data locality) and *autotuning* (automatic adaptation of the code to perform best according to processed data and used hardware).

The thesis is a collection of papers. It contains the commentary part, describing

the main contribution of the papers, summarizing their results, and putting them in the context of the state-of-the-art. The collection contains ten journal papers and four conference or workshop papers. The papers often span across multiple parts of the thesis: for example, they focus on GPU acceleration used in an approximative model or describe general optimization methods on application from cryo-electron microscopy. Except for three papers, the author of this thesis is the first or corresponding author of the papers included in the collection.

Contents

Part I

Commentary

Chapter 1

Introduction

When the computation demands of some application exceed the capabilities of hardware we can use for it, we can employ multiple strategies to decrease the demands. In multiple cases, we can formulate an algorithm, that solves the same problem, but in a better complexity class. In this thesis, we focus on cases when it is not possible, or the algorithm in a better complexity class is simply unknown. In such cases, there are still multiple ways of making the application less computationally demanding. In the case of scientific computing, we can attack the computational demands by changing a mathematical model. In some cases, a mathematically different model can be formulated, which sacrifices some precision for better computational speed (which can lead to better time complexity, or just reduce the amount of computations by a constant factor). We can also accelerate the implementation using specialized hardware, such as GPU processors, to speed up the application by a constant factor and get better power efficiency. In this thesis, we attack the computational demands at multiple levels: by changing the mathematical formulation of the model, by optimization of a source code, and by accelerating implementation on GPUs. Furthermore, we generalize some approaches to GPU acceleration and introduce their automation. The automation decreases the complexity of GPU programming while improving the performance portability of GPU-accelerated codes (i. e., the code performs well after input or hardware change from those used to optimize the code).

When dealing with the computationally demanding application, we apply the top-down approach, considering modification of the computational model first. In Chapter 2, we describe two examples of such an approach. First, we focus on modeling transport processes in proteins when a small molecule is transported via a protein's channel or tunnel into a position of our interest. Understanding those processes is of great importance in many fundamental biochemical processes as well as in various applications, such as drug design [63] or protein engineering [22]. Typically, the molecular system is simulated in time by molecular dynamics [101, 105], which requires enormous computational power if the transportation seldom occurs, as long evolution of the molecular system has to be simulated. Our contribution to this field lies in replacing molecular dynamics by searching for an energetically efficient path of the transported molecule (practically removing the time dimension from the simulation). The path search algorithm needs to pass a

many-dimensional continuous space of molecular movements and conformational changes. We introduced a novel algorithm, combining the search method with mathematical optimization based on molecular docking [108], which is modified to search for energetically favorable positions under constraints required by the search algorithm [29, 110].

The second application of model modification is haptic deformation modeling. Haptic rendering requires very high refresh rates for a smooth haptic perception (typically around 1 kHz). Simulation of human tissues is an important application, usable, e. g., in surgical simulators [10]. The deformable bodies, such as human organs, have complex geometry and display nonlinear responses [66]. The combination of high refresh rates required by haptic devices and computationally demanding models, such as nonlinear deformation of bodies, is challenging. We address this problem with a precomputation-interpolation scheme. Instead of computing the model in its present state in each iteration of the haptic loop, we pre-compute an area of close deformations of the deformable body at a lower refresh rate and interpolate from the pre-computed data at a refresh rate required by the haptic device. The close deformations can be computed independently so that the algorithm can scale to a high number of processors easily [36, 79].

Alongside papers [29, 36, 79, 110] included in this collection, we have also introduced a web interface for CaverDock [112], python interface [109], use it for massive screening [83, 84], sketch a new version of CaverDock including limited receptor flexibility [69], and improved haptic rendering by RBF interpolation [40].

Whether a model modification can speed up the computation or not, we can independently decrease the computational costs by optimizing the code or accelerating the application's bottlenecks on GPU processors. Contemporary GPUs have an order of magnitude higher arithmetic power than multi-core CPUs. Therefore, GPU acceleration brings constant-fold speedup. We demonstrate GPU acceleration in two application areas in Chapter 3: molecular modeling and cryo-electron microscopy. We have accelerated molecular map generation for AutoDock force field [68] by carefully fitting the computation of constant and distant-dependent dielectric to properties of GPU architecture [75]. Furthermore, we have improved state-of-the-art implementation of long-range molecular forces computation [95] by replacing grid z-dimension slicing with thread coarsening optimization technique [37]. We have also accelerated molecular clustering by introducing accelerated computation of dRMSD similarity metric [38], where we also show the limitations of the GPU cache system and formulate rules on how to use caches efficiently.

We also focused on the acceleration of the image processing algorithm from the area of cryo-electron microscopy. Cryo-electron microscopy is a popular method for studying the structure of biological specimens in their native environment: water. The method suffers from a very low signal-to-noise ratio. Therefore, a high amount of computations is needed to reach near-atomic resolution [44]. We focused on GPU acceleration of 3D reconstruction [2]. The original implementation is difficult to parallelize efficiently due to poor memory locality and overlapping memory writes. We have introduced a novel GPU-friendly algorithm, which uses alternative mapping of threads to input and output arrays, removing race conditions and improving memory locality [99]. This novel approach brings an order of magnitude speedup compared to parallel CPU implementations and a several-fold speedup compared to previous GPU implementations. We also focused on

one of the first steps in the cryo-electron microscopy pipeline: the movie alignment. We introduced FlexAlign [96]. This work demonstrates a complex GPU-accelerated application, connecting FFT transformation with auto-tuned settings, GPU-accelerated kernels for image correlation analysis, and CPU-based mathematical optimization.

Alongside papers [37, 38, 75, 96, 99] included in the collection, we have also accelerated RMSD computation for molecular metadynamics [34], element subroutines for haptic deformation modeling described in Chapter 2 [31, 35], movie alignment using deep-NN [53] and deformation fields of macromolecules [45–47] for cryo-electron microscopy.

Some optimization approaches are usable for multiple algorithms spanning different application domains. We focus on kernel fusion and autotuning in Chapter 4. Kernel fusion is a method of connecting multiple parallel functions (kernels) into one [102, 114, 117]. It allows to improve data locality, increase parallelism, or eliminate common subexpressions. We have used kernel fusion to improve the speed of element subroutines in deformation modeling [35] and investigated the effect of different methods of kernel fusion optimization targetting GPUs, CPUs, and Xeon Phis [30]. Furthermore, we have investigated automation of the kernel fusion: we proposed a source-to-source compiler, allowing to fuse any CUDA kernels performing parallel map [39], reduce, and possibly nested parallel map and reduce [33]. We demonstrated that with automatic fusion on typical sequences of BLAS operations, it is possible to outperform vendor-optimized BLAS library more than 2× [33].

Most GPU-accelerated applications are implemented in CUDA (supported on NVIDIA architectures and x86 processors) or OpenCL (supported over a broad range of GPUs, CPUs, and FPGAs). Although it is possible to implement a portable code, the performance portability is poor across different hardware architectures [56] or problem inpus [42]. Autotuning allows optimizing the application's tuning parameters (properties influencing the application performance) to perform the execution more efficiently and hence solve the performance portability issue. We have introduced an autotuning framework called Kernel Tuning Toolkit, allowing us to tune CUDA and OpenCL source code [37]. Moreover, we demonstrated that this tuning can be performed dynamically during program execution [82]. We also experimentally showed that autotuning solves the performance portability problem, as the unified codebase was tuned to near-peak performance for multiple different GPU architectures [82]. We further focused on improving the autotuning process in terms of scheduling and tuning space search efficiency. We proposed a model allowing us to predict the number of tuning space search iterations [74], which is necessary for deciding whether autotuning should be executed under a limited time budget. Furthermore, we proposed a novel method for tuning space search [32], which utilizes hardware performance counters in order to analyze bottlenecks of a tuned code and decide how to change tuning parameters in order to speed it up. As this method understands the performance problems of the tuned code, it outperforms both mathematic optimization-based solutions and end-to-end machine learning solutions. Last but not least, we showed that autotuning can also be applied to the size of data selection in the case of Fast Fourier transform [98]. When the application allows for padding or cropping the data, the proper selection of its size can improve the speed of FFT more than 6×. The most efficient data size depends on the FFT implementation and GPU model. Therefore, autotuning with heuristic pruning is used to search for the best data size empirically.

Alongside papers [32, 37, 74, 82, 98] included in this collection, we have also introduced Kernel Tuning Toolkit 2.0 [81], a benchmark for autotuning frameworks [107], and a framework integrating autotuning into image-processing pipelines [97].

1.1 Papers in the Collection

This thesis is a collection of papers. In this section, we list the papers used in this thesis and explicitly describe the thesis author's contribution.

- [29] *Jiří Filipovič*, Ondřej Vávra, Jan Plhák, David Bednář, Sérgio M. Marques, Jan Brezovský, Luděk Matyska, Jiří Damborský. CaverDock: A Novel Method for the Fast Analysis of Ligand Transport. In *IEEE/ACM Transactions on Computational Biology and Bioinformatics, Volume 17, Issue 5*, 2019. [Author's contribution: **First author (shared)**, 35 %: method design and implementation, writing.]

- [110] Ondřej Vávra, *Jiří Filipovič*, Jan Plhák, David Bednář, Sérgio M. Marques, Jan Brezovský, Jan Štourač, Luděk Matyska, Jiří Damborský. CaverDock: a molecular docking-based tool to analyse ligand transport through protein tunnels and channels. In *Bioinformatics, Volume 35, Issue 23*, 2019. [Author's contribution: **First author (shared)**, 35 %: method design and implementation, writing.]

- [79] Igor Peterlík, *Jiří Filipovič*. Distributed Construction of Configuration Spaces for Real-Time Haptic Deformation Modeling. In *IEEE Transactions on Industrial Electronics, Volume 58, Issue 8*, 2011. [Author's contribution: 40 %: discussion, some implementation, some writing, experiments.]

- [75] Marek Olšák, *Jiří Filipovič*, Martin Prokop. FastGrid – the Accelerated Auto-Grid Potential Maps Generation for Molecular Docking. In *Computing and Informatics, Volume 29, Issue 6a*, 2010. [Author's contribution: 20 %: discussion, some implementation, writing.]

- [37] *Jiří Filipovič*, Filip Petrovič, Siegfried Benkner. Autotuning of OpenCL Kernels with Global Optimizations. In *ANDARE '17: 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems*, 2017. [Author's contribution: **First author**, 35 %: discussion, method, some implementation, writing.]

- [38] *Jiří Filipovič*, Jan Plhák, David Střelák. Acceleration of dRMSD Calculation and Efficient Usage of GPU Caches. In *International Conference on High Performance Computing & Simulation (HPCS)*, 2015. [Author's contribution: **First author**, 40 %: discussion, method, experiments, writing.]

- [99] David Střelák, Carlos Óscar S. Sorzano, José María Carazo, *Jiří Filipovič*. A GPU acceleration of 3-D Fourier reconstruction in cryo-EM. In *The International Journal of High Performance Computing Applications, Volume 33, Issue 5*, 2019. [Author's contribution: **Principal investigator**, 30 %: method, some implementation, experiments, writing.]

- [96] David Střelák, *Jiří Filipovič*, Amaya Jimenénez-Moreno, José María Carazo, Carlos Óscar S. Sorzano. FlexAlign: An Accurate and Fast Algorithm for Movie Alignment in Cryo-Electron Microscopy, In *Electronics, Volume 9, Issue 6*, 2020. [Author's contribution: 20 %: some method design, some experiments, some writing.]

- [30] *Jiří Filipovič*, Siegfried Benkner. OpenCL Kernel Fusion for GPU, Xeon Phi and CPU. In *27th International Symposium on Computer Architecture and High Performance Computing*, 2015. [Author's contribution: **First author**, 90 %: discussion, method, implementation, some writing.]

- [33] *Jiří Filipovič*, Matúš Madzin, Jan Fousek, Luděk Matyska. Optimizing CUDA code by kernel fusion: application on BLAS. In *Journal of Supercomputing, Volume 71, Issue 10*, 2015. [Author's contribution: **First author and Principal investigator**, 50 %: discussion, method, some implementation, experiments, writing.]

- [82] Filip Petrovič, David Střelák, Jana Hozzová, Jaroslav Oĺha, Richard Trembecký, Siegfried Benkner, *Jiří Filipovič*. A benchmark set of highly-efficient CUDA and OpenCL kernels and its dynamic autotuning with Kernel Tuning Toolkit. In *Future Generations Computer Systems, Volume 208*, 2020. [Author's contribution: **Principal investigator**, 20 %: discussion, method, some implementation, experiments, writing.]

- [74] Jaroslav Oĺha, Jana Hozzová, Jan Fousek, *Jiří Filipovič*. Exploiting historical data: Pruning autotuning spaces and estimating the number of tuning steps. In *Concurrency and Computation: Practice and Experience, Volume 32*, 2020. [Author's contribution: **Principal investigator**, 30 %: discussion, method, experiments, writing.]

- [32] *Jiří Filipovič*, Jana Hozzová, Amin Nezarat, Jaroslav Oĺha, Filip Petrovič. Using hardware performance counters to speed up autotuning convergence on GPUs In *Journal of Parallel and Distributed Computing, Volume 160*, 2022. [Author's contribution: **First author, Principal investigator**, 50 %: discussion, method, some implementation, some experiments, writing.]

- [98] David Střelák, *Jiří Filipovič*. Performance analysis and autotuning setup of the cuFFT library. In *ANDARE '18: 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems*, 2018. [Author's contribution: **Principal investigator**, 20 %: discussion, some method, some experiments, some writing.]

Chapter 2

Approximative Modelling

In this chapter, we focus on methods, which decrease computational demands by utilizing an alternative model of the computation. We select a model, which is computationally less demanding, resulting in slightly different, but still valuable, results. Section 2.1 is based on papers [29, 110], and describes a novel approach to model transport processes in proteins. Section 2.2 is based on paper [79] and introduces a precomputation-interpolation scheme allowing haptic modeling of deformable bodies with complex mathematic models.

2.1 Modelling Transport Processes in Proteins

Understanding *protein-ligand interactions* (interaction of a small molecule – a ligand, with a big biomolecule – a protein) is of great importance in many fundamental biochemical processes as well as in various applications. For example, studying a ligand that may inhibit protein function allowing a virus to attack a cell, or to designing inhibitors blocking tunnels and channels is considered to be a new paradigm in drug design [63]. The ligand interacts with the protein in its *active* or *binding site* – the functional site of a protein. Many proteins have binding sites buried inside their cores, which implies that a ligand must traverse through a *tunnel* before it can bind to the functional site in the protein. In such cases, we need to analyze whether the ligand is likely to pass through the tunnel into the protein core.

All chemical systems, such as the proteins interacting with ligands, follow the second law of thermodynamics: they tend to minimize their potential energy. In practice, the most probable *conformation* of the molecules (i. e., spatial position of their atoms) is the one with the lowest potential energy. However, it is also possible for molecules to make a transition from one local minimum to another, depending on system temperature and height of the energetic barrier between minima – the smaller the energetic barrier, the more probable the transition.

The ligand binding to a protein's binding site is usually computed by *molecular docking*. A molecular docking is in principle a mathematical optimization algorithm. The interactions between ligand and protein are approximated by a *force field*, and the docking algorithm traverses the conformation space of the protein-ligand complex and searches

for energetic minima of the force field [26, 68, 104, 108]. The result of the molecular docking is the structure of the protein-ligand complexes together with an estimation of the respective free energy of binding. Thus, the users can learn which ligand binds with the lowest energy or study the orientation of a ligand in a protein active site. However, molecular docking computes only the lowest energy positions of a ligand within some region of a protein, and thus it is not suitable for the study of the ligand transport through a protein tunnel.

The access tunnel can be studied by *geometry-based* approaches [19, 93, 118]. The geometry-based approach does not consider any chemical forces – the accessibility is determined from the tunnel geometry only.

The *molecular dynamics* (MD) uses an empirical force field to model the physical properties of the atoms and their interactions in time. The ligand passage through a tunnel can be studied by well-established software tools for MD, such as Amber [91] or Gromacs [1]. However, it is not practical to model the transportation of a ligand through a tunnel with classical MD, as the simulation time is often extremely long. The metadynamics [13] and steered molecular dynamics [101] are enhanced techniques. Metadynamics biases molecular energy towards unexplored configurations (metadynamics), whereas steered-MD applies external force to the ligand to pass a tunnel quickly (steered MD). Although it is possible to accelerate passing the tunnel with such methods [105], their settings require a domain expert with good knowledge of the studied protein and are not suitable for automatic analysis of the huge amount of ligands (e. g., potential drugs). Moreover, those methods are still more computationally demanding than geometrical approaches.

2.1.1 Contribution

One possible approach to analyze a ligand's ability to pass tunnel in protein is a computation of the ligand's path with minimal energy (i. e., the path which crosses the lowest energetic barriers possible, without simulating ligand in time as it is done by MD). However, as a ligand is a flexible body, its movement has typically tens of degrees of freedom (position, orientation, and torsions), classical path-finding algorithms are not practical: they either discretize the space, which quickly exceed memory capacity due to high dimensionality, or they generate random positions of ligand and connect them, which is not usable in narrow parts of the tunnel (as most of the random ligand's positions are unfeasible). The main idea behind our method is to (i) generate suitable ligand snapshots along the tunnel by molecular docking (the mathematical optimization places ligand in feasible positions) and (ii) connect the snapshots (possibly by generating new ones) into continuous trajectory. Therefore, it combines mathematical optimization and a path-searching algorithm, creating new snapshots on demand, if the path representing continuous movement cannot be found. The newly generated snapshots are located in local minima (due to molecular docking), satisfying constraints (generated by the search method). The method is presented in papers [29, 110].

CaverDock uses geometrical information about the tunnel as an input. Before CaverDock execution, we need to compute the tunnel by Caver [19] or a similar tool. The tunnel is defined as a sequence of spheres in 3D space. Then, CaverDock discretizes the

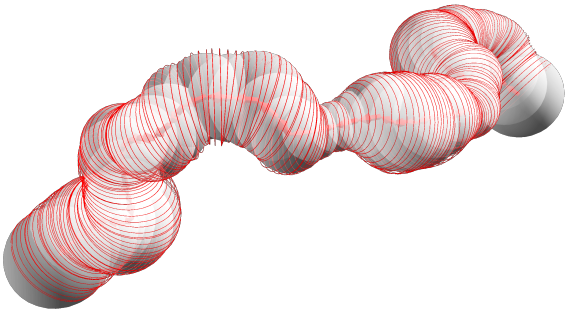


Figure 2.1: Discretization of a tunnel in the native toluene/o-xylene monooxygenase hydroxylase. The red circles represent the discs, the red arrows represent the tunnel direction and the grey spheres represent the tunnel obtained from Caver [19].

tunnel: it cuts the tunnel finely into slices of upper-bound thickness [29]. Those slices are represented as discs in 3D space. See Figure 2.1 for an illustration of the discretized tunnel.

CaverDock internally uses a molecular docking algorithm and force field derived from AutoDock Vina [108]. AutoDock Vina's docking is a stochastic mathematical optimization method that takes positions and types of all the atoms in the system (i. e., both the ligand and the receptor) and optimizes the position of the ligand's atoms in order to minimize the empirical energy (a function of atoms' positions and types). Molecular docking is able to find one or more suitable ligand snapshots (spatial position of all its atoms), but users have limited control over the position where the ligand is docked. As CaverDock uses docking to generate a continuous trajectory of a ligand, it implements restraints into AutoDock Vina force field, which allows for better control of both the position of the ligand in the tunnel (ligand is snapped to the disc) and the pattern restricting the acceptable position of ligand's atoms (ligand is docked into the vicinity of its previous position).

A pre-selected ligand's atom (typically an atom in the center of the ligand) can be snapped to a disc: *position restraint*. By applying the position restraint iteratively, a disc by disc, CaverDock generates a non-continuous trajectory of the ligand: with this step-by-step approach, the ligand can change its rotation or conformation freely between discs and even miss some obstacles, see Figure 2.2 for illustration. Such a non-continuous trajectory is called a *lower-bound trajectory*, as it leads to an optimistic energy profile.

The second part of CaverDock execution attempts to generate a continuous trajectory of the ligand by restraining other atoms of the ligand: it forces atoms' positions to stay in the vicinity of their position at the previous disc: *pattern restraint*, see Figure 2.3. This ensures the continuity of the ligand's movement while still taking into account the position restraints and the energy taken from molecular docking.

To obtain energy-efficient continuous movement, we need to search the space of possible ligand atoms' trajectories. We choose to minimize either the maximal energy of any snapshot in the trajectory or the cumulative energetic barrier of the whole trajectory.

Seeking for the optimal trajectory is computationally unfeasible even for short tunnels and simple ligands due to the high dimensionality of the search space. Therefore, a

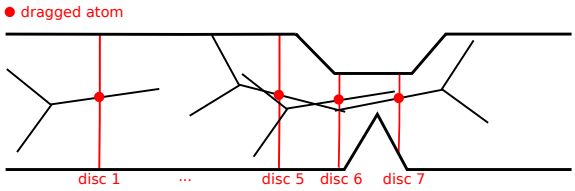


Figure 2.2: Schematic 2D view of Y-shaped ligand traversing a tunnel. The selected ligand's atom (depicted as a red circle) is placed onto consecutive disks (red lines). As no contiguous movement of the ligand is required, the ligand flips between disks 6 and 7, thus the small geometrical bottleneck between those disks is not detected.

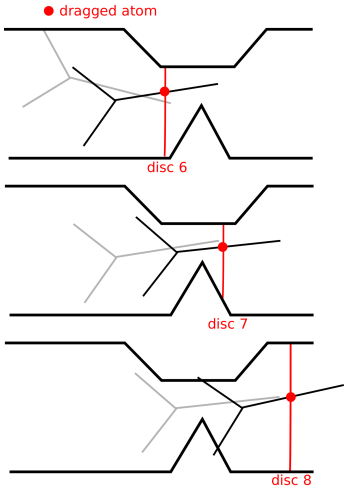


Figure 2.3: Schematic 2D view of a ligand traversing a tunnel. The ligand is depicted in black, its previous position used as a pattern restraint is shown in grey. Restricting the movement of atoms causes the geometrical bottleneck between discs 6 and 7 to be detected when the ligand passes from disc 7 to disc 8, as can be seen in the last figure.

heuristic search in this optimization space is used. First, with a given pattern and position restraint, only a few local minima are computed by molecular docking, so not all possible snapshots are considered. Second, to avoid getting stuck in the narrow part of the tunnel in suboptimal rotation or conformation of the ligand, CaverDock combines forward and backward movement of the ligand. When a strong energetic barrier is reached at some disc, a different snapshot of the ligand at the same disc is selected and the ligand is moved backward. Then, several scenarios can happen: (i) the previously computed forward and backward trajectories converge, (ii) the ligand reaches the first disc, replacing the original forward trajectory, or (iii) the ligand hits another energetic barrier, which is stronger than the one triggering the backward movement, and the backward trajectory is discarded. Finally, CaverDock continues from the disc where the backward movement is triggered. The trajectory obtained this way is called the *upper-bound trajectory*, as it is the result of the search heuristic: the optimal trajectory may have a better energetic profile than the

trajectory found by CaverDock, so it lies between upper- and lower-bound.

The result of CaverDock is continuous movement of the ligand, going from the active site to the tunnel entrance (or vice versa), and its energetic profile. Compared to MD-based methods, no expert user is needed to configure CaverDock, and the trajectory can be easily studied (MD simulates the stochastic movement of the ligand in time, so the trajectory typically contains a lot of movements not related to moving ligand from/to the tunnel or going back and forth in some part of the tunnel), making CaverDock ideal tool for virtual screening. Compared to MoMA-LigPath [24], CaverDock generates energetic profiles and not only continuous trajectories. Compared to SLITHER [59], CaverDock generates a continuous trajectory without gaps. We have evaluated CaverDock on a set of biologically relevant molecules [110], showing it is a robust tool that finds ligands trajectory in cases its competitors SLITHER [59] and MoMA-LigPath [24] fail. Furthermore, it successfully observes blockages in tunnels in multiple variants of haloalkane-dehalogenase LinB, agreeing with results observed in [14].

In our following work, we show CaverDock can be used for massive screening of FDA-approved drugs against pharmacologically relevant targets [83] and against spike-glycoprotein of the new SARS-CoV-2 virus [84]. The CaverDock calculations are also available to the broad community via the CAVER web server [112].

2.2 Haptic Deformation Models

Haptic rendering is one of the most advanced techniques in the area of human-computer interaction. Besides vision, haptics represent another important perception necessary for most of human activities. Therefore, human-computer interfaces employing haptic devices provide new approaches to understanding many phenomena, e. g., in computer simulation of physical processes.

We focus on the area of haptic rendering which enables a user to touch, feel, and manipulate virtual objects through a haptic interface [11, 58]. Similarly, as in the case of visual rendering, haptic rendering can be a very efficient tool for studying various phenomena, e. g., to display a deformed body under applied forces. One of the basic issues in the area of haptic rendering is caused by the high sensitivity of human touch. When compared to visual perception where a refresh frequency of 60 Hz is sufficient to provide the illusion of a smooth motion, the refresh rate needed in haptics exceeds 1 kHz. As the force often depends on the global behavior of the underlying mathematical object, only simple models can be considered for the case when the force is computed directly in the haptic loop. However, usually complex models showing, e. g., non-linear behavior are of great interest.

There have been several attempts to address the high-frequency issue, such as simplification of the underlying models [15, 85] or employing some precomputations before the real interaction occurs [9, 20]. We focus mainly on the design of the *surgical simulators* (see, e. g., [10]). Soft tissue simulations require a rather computationally demanding mathematical model: the deformable bodies (e. g., organs) have usually complex geometry and the behavior displays non-linear response [66]).

2.2.1 Contribution

We introduced a method suitable for the stable haptic rendering of complex objects with non-linear behavior in our papers [36, 79]. The method builds on the previous precomputation method introduced in [80]. The main idea behind the approach is to profit from using a high-performance computing environment such as a grid to overcome the basic issue of the haptic rendering given by the high refresh rate.

The mathematical representation of the deformable body works with its discretized finite mesh. The deformation is solved numerically by *finite element method*. The method results in a non-linear system $\mathbf{K}(\mathbf{u}) = \mathbf{F}$ of $3N$ algebraic equations where N is the total number of the nodes in the FE mesh. In order to solve the non-linear system, *boundary conditions* are imposed, so the solution is unique. First, the homogeneous conditions are defined by choosing a set of surface nodes with zero displacement. Second, the non-homogeneous conditions are introduced. In the current implementation, the *displacement-driven point interaction* is used to couple the haptic interaction and deformation model [78]. The interaction point is snaped to a node of the finite mesh and is driven by the haptic device: the position of the haptic interaction point determines the displacement of the snapped node, resulting in a deformation of the entire body and a corresponding response force acting in the snapped node and hence haptic device. Mathematically, the prescribed displacement of the snapped node is represented by the non-homogeneous Dirichlet boundary condition, which is implemented using Lagrange multipliers [80]. The non-linear system given above is solved using the Newton-Raphson method: in each step, a linearized system based on the spatial derivative of the non-linear mapping $\mathbf{K}(\mathbf{u})$ is assembled, both the homogeneous and non-homogeneous Dirichlet conditions are applied using Lagrange multipliers and the augmented system is solved to obtain a new estimation of the unknown vector of the response force and nodal displacements. The process is iterated until the convergence is achieved that is $\|\mathbf{K}(\mathbf{u}) - \mathbf{F}\| < \varepsilon$ for some small ε.

The system assembly and solving is highly computationally demanding and cannot be solved at the frequency required by the haptic interface. Therefore, we have introduced a precomputation method, which precomputes close deformation on a regular grid. The precomputation happens in time of the haptic simulation, so only a part of the configuration space, which can be actually visited, is precomputed. The main idea of the method is as follows: after specifying the parameters of the simulation (material coefficients, nodes fixed in space, and snapped node), the interaction starts being composed of two main processes: construction of new configurations and interpolation of the configurations. The computationally expensive constructions are performed in a distributed environment, whereas the interpolation is simple enough to be calculated in the haptic loop which is executed on a single PC attached to the haptic device. Apparently, the main task of the algorithm is to ensure that the data required by the interpolation method are constructed on time, so it can be used when needed.

The implementation of our method is based on a client-server scheme. The client is represented by a *haptic driver* and *scheduler*, whereas the server is represented by a distributed pool of *solvers* running as independent processes on remote nodes connected to the client by a network. The haptic driver executes the haptic loop running at the

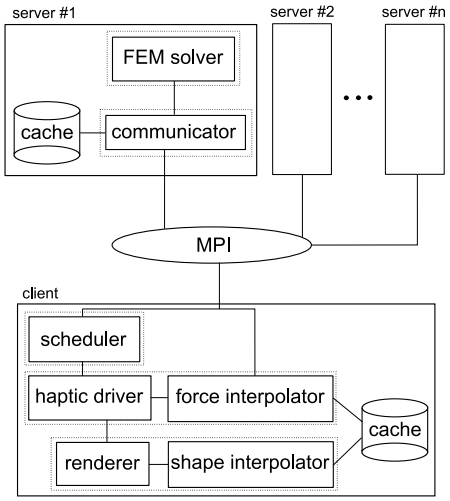


Figure 2.4: Scheme of the algorithm components (solid-outlined boxes) grouped in threads (dashed-outlined boxes).

high frequency of 1,000 Hz. It is responsible for haptic rendering and communication with the haptic device. The scheduler running at the lower frequency of 100 Hz is responsible for constructing the configuration space that is needed by the haptic driver. During the simulation, it schedules the computation of the configurations among the solvers according to the motion of the haptic interaction point. Finally, each solver works as an event-driven process: it is waiting for the message from the scheduler containing the transition which is to be computed. Immediately after the message is received, it executes an iterative numerical procedure based on the Newton-Raphson method. The architecture of the proposed application is depicted in Figure 2.4. The illustration of the precomputation in relation to the haptic device movement is given in Fugure 2.5.

Quantity	SV M_1	SV M_2	MR M_1	MR M_2
$t_{conf}[s]$	0.42	3.15	1.47	10.9
$v_{max}[mm/s]$	24.1	3.1	6.8	0.92

Table 2.1: Table showing the time needed for computation of one deformation (t_{conf}), and maximum allowed speed of HIP during the on-line construction (v_{max}), i. e., observing a new part of deformation space.

Obviously, the speed of the haptic device is limited by the speed of precomputation (as the haptic interaction point cannot leave the precomputed area). We have shown that with dozens of computing nodes, the limitation of the speed is sufficient [79] – see Table 2.1. The results are evaluated for two models of liver: M_1 is a coarse mesh composed of 1777 elements, and fine mesh M_2 includes 10280 elements. Two materials were sim-

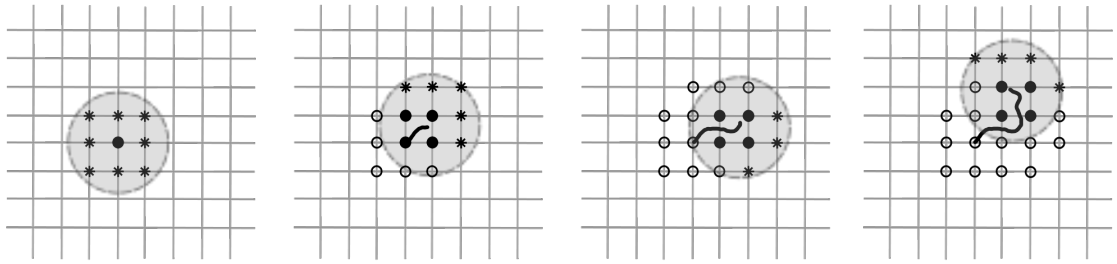


Figure 2.5: Four snapshots of the online precomputations. The trajectory of the haptic interaction point is visualized by the thick black curve, its vicinity is depicted by the gray circle, and configurations are denoted as follows: full circles show the configurations computed before and actually used for the interpolation, empty circles show the configurations computed before and currently unused (cached), and stars show configurations being actually computed to be potentially used after cell switching.

ulated: Mooney-Rivlin (MR) and StVenant-Kirchhoff. The speed of the haptic interface is acceptable for both materials using the coarse model, although computation of one deformation can be more than one second, i. e., three orders of magnitude slower than the haptic loop.

Additionally to the papers included in this collection, we also introduced RBF interpolation and extrapolation into the scheme [40], which relaxes the speed restrictions even more. We have also demonstrated that kernel fusion and change of parallel granularity improve the speed of the system of equation assembly [35] and demonstrated automation of fusions by the compiler presented in Section 4.1 in [31].

Chapter 3

Applications of GPU Acceleration

Contemporary GPUs provide an order of magnitude higher arithmetic power and multiple times higher memory bandwidth compared to conventional CPUs. However, as GPUs implement massive parallelism, execute threads in lock-step mode, and have very limited caching capabilities (L1 caches are typically for read-only data), using their arithmetic power and memory bandwidth efficiently is challenging. Therefore, algorithms typically have to be carefully optimized for GPUs or even re-designed from scratch to fit GPU architecture well.

The GPUs were successfully used to accelerate software in virtually any scientific computing domain. Essential HPC libraries, such as vendor-provided cuBLAS, cuFFT, and their open-source alternatives MAGMA [4], clBLAST [70], MPFFT [61], allows replacement of their CPU-based counterparts and acceleration of software using those libraries without need of invasive changes of a source code. However, in many cases, specialized algorithms whose performance does not rely on those libraries must be accelerated to speed up the software.

In this chapter, we focus on the application of GPU acceleration in scientific computing. Section 3.1 is based on papers [37, 38, 75] and describes the acceleration of two methods relevant in computational chemistry. Section 3.2, which is based on papers [96, 99], describes the acceleration of two important image processing steps in cryo-electron microscopy.

3.1 GPU Acceleration in Computational Chemistry

Computational chemistry employs computer simulation to solve chemical problems, such as the determination of molecular structure, simulation of molecular interaction, conformational changes, etc. Compared to the wet-lab experiment, the computer simulation is typically cheaper and safer. Moreover, it allows us to observe some phenomena that cannot be directly observed experimentally, such as conformational changes of large molecules or protein-ligand or protein-protein interactions.

Here, we focus on *molecular mechanics*, which simulates large molecular systems as balls and springs, which empirically determined and parametrized interactions. Although molecular mechanics have lower precision compared to methods based on quantum chem-

istry (i. e., solving or approximating Schrödinger equation), it is also less computationally demanding, allowing simulation of large biomolecules. The molecular mechanics is an instance of *n-body* problem; therefore, its computational time scales at worst quadratically with respect to the number of atoms. The computational demands of molecular mechanics become challenging when long evolvement, large molecular systems, or many different systems are simulated. For example, an all-atom simulation of HIV-1 capsid was performed in 2017, using 3880 GPU-accelerated nodes of the TITAN supercomputer [77]. Nowadays, the GPU accelerated molecular simulation is available to the broad community in probably the most popular programs for molecular dynamics based on molecular mechanics, AMBER [91] and GROMACS [1].

Another computationally demanding molecular mechanic task is *molecular docking* [68, 108]. Molecular docking is a method for the prediction of a position and an orientation of one molecule to another when bound to each other to form a stable complex. One of the categories of molecular docking, which is important in computer-aided drug design, is protein-ligand docking, where a small molecule of a ligand (putative drug) is docked into a binding site of a protein. Calculation of *interaction energy* between the protein and the ligand is the most computationally demanding part of docking algorithms. The energy is typically pre-computed in a regular grid and then interpolated during searching for the most favorable ligand position. Although molecular docking is generally less computationally demanding than molecular dynamics, it is typically used to screen large amounts of molecules, significantly increasing computational demands [67].

3.1.1 Contribution

We have extended state-of-the-art in GPU-accelerated molecular mechanics and molecular modeling in several directions. First, we focused on molecular docking. We have extended GPU-accelerated *electrostatic potential map* generation introduced by Stone et al. [95] by adding *distance dependent dielectrics* and integrating GPU-accelerated long-range interactions with CPU-computed short-range interactions [75]. We have also further accelerated electrostatic map generation by introducing thread coarsening in z-dimension [37]. Second, we have introduced GPU-accelerated computation of dRMSD, a distance metric for molecules, used in their clustering [38].

Docking programs, such as AutoDock [68], precompute energetic potential around a protein into a regular grid by a program AutoGrid. AutoGrid pre-computes multiple energetic terms at a 3D grid surrounding a molecule: *electrostatic* potential, *van der Waals* potential, and *hydrogen bonds*. In [95], a method for precomputing electrostatic potential on GPU was presented. The most computationally extensive part of the algorithm is the computation of Euclidean distance between atoms and points of the 3D grid. The method slices the 3D grid to the 2D plate and pre-computes squared distances in z-dimension at CPU, reducing the amount of arithmetic operations performed by GPU. Our contribution here is the implementation of FastGrid [75]. FastGrid introduces hybrid CPU-GPU computation. On the GPU side, a long-range electrostatic potential is computed. We extended original constant dielectric [95] potential computation by option to compute distance-dependent dielectric. The distance-dependent dielectric approximates the pro-

gressive decline of the electrostatic potential due to ions and water dipoles in the natural environment. The computation of distance-dependent dielectric requires more arithmetic operations than the original constant dielectric. Therefore, we used a precomputed table of the distance-dependent dielectric term cached in texture memory and implemented an on-the-fly computation of the distance-dependent term. Depending on cache efficiency, the algorithm uses either pre-computed data or on-the-fly computation [75]. Moreover, we have implemented a hybrid CPU-GPU implementation, where short-range interactions are computed on the CPU. As the long-range interactions have higher asymptotic complexity, GPU performance is crucial for large molecules. Compared to the original single-threaded AutoGrid computation, we reached more than 400× speedup using 4 CPU cores and GeForce GTX 280.

We further analyzed the constant dielectric computation introduced in [95] and came up with a new optimization for this algorithm. Instead of precomputation of z-distance on the CPU side, we serialize several iterations over z-dimension in GPU threads [37]. Such implementation uses a slightly higher number of arithmetic operations than [95], but improves strong scaling by using more threads simultaneously and reduces the overhead of GPU-CPU communication. With the new optimization, we reached a speedup of factor 1.14× at GeForce GTX 1070 over the state-of-the-art.

RMSD (root-mean-square deviation) is used in computational chemistry to compare different structures of a molecule. It is used, for example, in *clustering* of similar molecules or in molecular dynamics to quantify changes in simulated systems. We focused on dRMSD calculation, originally introduced in [106]. We introduced parallel GPU-accelerated implementation of dRMSD computation in [38]. The implementation parallelizes a nested loop going over atoms, synchronizing memory access to load batches of atoms into shared memory. Furthermore, we increase register locality by thread-coarsening (see, e. g., [111]). By careful tuning of the algorithm, we show that it is possible to rely on GPU caching capabilities and reach performance close to the version using explicit caching in shared memory. Compared to parallel CPU implementation in ClusCo [51], we reached 62.7× speedup of dRMSD calculation, yielding 13.4× faster clustering.

3.2 GPU Acceleration in cryo-Electron Microscopy

Cryo-electron microscopy (cryo-EM) is a popular method for studying a *structure* of biological specimens, such as proteins or larger particles, e. g., viruses. In contrast to X-ray crystallography, the specimen is studied in vitreous ice at cryogenic temperatures, which allows it to preserve the same conformation as in the native environment. Compared to nuclear magnetic resonance, cryo-EM allows studying larger structures, making it a superior method in many use cases. In recent years, rapid development in cryo-EM allowed us to study specimens at near-atomic resolution [44], resulting in the identification of cryo-EM as the method of the year by Nature Methods in 2015 and granting Nobel price in Chemistry for cryo-EM pioneers in 2017.

Due to a low signal-to-noise ratio and unknown orientation of frozen specimens, obtaining a 3D structure from cryo-EM heavily depends on *image processing*. The image

processing pipeline of the Cryo-EM project might be very complicated. However, it always contains certain computationally demanding steps. Probably the most notable ones are *movie alignment*, *2D classification*, and *3D refinement*. In this section, we focus on GPU acceleration of the movie alignment and 3D refinement.

To obtain a single frame, a beam of electrons is fired against the sample. After passing the sample, the beam is recorded by a direct electron detection camera. Since the electron beam causes radiation damage, the electron dose has to be very low. This, on the other hand, results in an extremely low signal-to-noise ratio. To get more signal, we can repeat the imaging several times using the same sample and then average the resulting frames. However, before averaging the frames, they have to be properly aligned as the sample moves during the acquisition [41]. This movement can be both global and local, and both types need to be corrected. There are multiple programs available to perform movie alignment. The Optical Flow [3] has high computational demands as it is not accelerated on GPUs. Relion [121] provides both local and global alignment but also lacks GPU-accelerated implementation. The same approach as in [121], but carefully optimized and accelerated, is implemented in MotionCorr2 [120]. Although MotionCorr2 is very fast, it is a closed-source implementation and does not provide some important data, such as per-pixel movement, required for particle polishing.

During the 3D reconstruction, a 3D volume is created from a large number of *2D projections* (images of the specimen). However, the orientations of projections are not known *a priori*. In order to determine the orientation of projections, we need to solve the inverse problem iteratively: creation of the 3D volume from projections, which is a nontrivial process due to noise in images, errors in orientation parameters, and the finite number of discrete parameters covering the projection space non-uniformly [76]. There are multiple approaches to the 3D reconstruction [2, 87, 94]. One of possible approaches is *direct Fourier reconstruction* [2], based on the central slice theorem [21, 54]: the 2D Fourier transform of the projection of the 3D object lies on the plane centered at the origin of the 3D Fourier transform (FT) of the object and preserves the same orientation as the projection. In order to reconstruct a 3D body from a given set of projections and their orientations, we need to insert the FT of all images into a 3D voxel array and then obtain the final 3D shape by performing inverse FT of the voxel array. The most computationally demanding part of the direct Fourier reconstruction is the 2D FT of input images and their insertion into the voxel array.

3.2.1 Contribution

We have introduced the program FlexAlign, a GPU-accelerated global and local movie alignment, in [96]. The performance of FlexAlign heavily depends on forward FT performance and the speed of data movements between CPU and GPU memory.

For the FT, we use vendor library cuFFT [73]. The performance of cuFFT heavily depends on the size of input data, which can be modified in cryo-EM: the microscope provides images in giver resolution; however, we can crop these images. When cropping only a small fraction of an image, the information loss is negligible, whereas FT speedup can be several-fold. We have proposed a novel auto-tuning approach [98], which searches

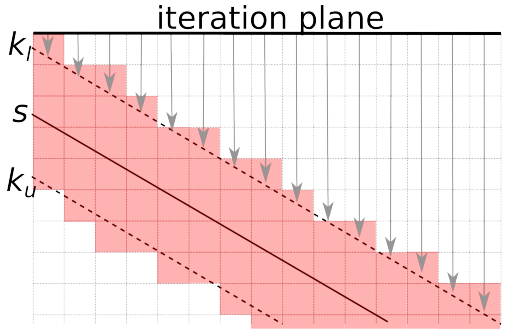


Figure 3.1: Schematic view of the iteration space in the cut of the 3D grid. The solid line represents a 2D sample, dashed lines represent the boundaries of an area affected by the interpolation window. Arrows show the computation of the initial iteration in the third dimension (i. e. dimension not iterated at the iteration plane). The updated voxels are emphasized.

for similar sizes of FT input, which leads to much higher FT speed (see Section 4.2 for more details). With this approach, we can get up to 6.9× faster FT computation.

The FlexAlign algorithm requires the computation of correlation for all pairs of images. As one movie easily exceeds the capacity of GPU memory, we have designed a pipelined batch processing of them: the images are uploaded to GPU in batches, such that multiple correlations are computed per image upload (inside the batch and among two batches). This approach allows for decreased PCI-E bandwidth, so the FlexAlign speed is determined by FT speed on mid-range GPUs.

A GPU acceleration of 3D Fourier reconstruction is described in [99]. To the best of our knowledge, all state-of-the-art GPU implementations of the 3D Fourier reconstruction use the scatter memory pattern [55, 60, 100, 119], which writes each pixel of the 2D projection into multiple voxels of the 3D space (multiple voxels are affected due to the interpolation). Although it is well known that scatter memory pattern is suboptimal when data accessed by multiple threads overlap, it is not straightforward to formulate a 3D Fourier reconstruction with a gather memory access pattern.

We introduced a novel approach to parallelization 3D Fourier reconstruction, which results in a gather memory access. With our parallel algorithm, the value of each voxel in the output 3D volume is computed by interpolating from multiple pixels of the 2D projection. It eliminates race conditions in writing into the 3D volume and improves memory locality as repeated memory accesses are moved from the 3D volume into the much smaller 2D projection. When implemented naively, the gather memory pattern raises complexity from $\mathcal{O}(in^2)$ to $\mathcal{O}(in^3)$, where i is the number of images insertions (number of images multiplied by the number of symmetries) and n is images' resolution. We introduced and concept of iteration plane: a 2D space of parallel threads, which computes only relevant voxels of their columns (see Figure 3.1). The iteration plane is always orthogonal to one of three axes x, y, z. The orthogonal axis is selected to maximize the projection of the 2D projection to the iteration plane. Then, for each pixel of the

iteration plane, a GPU thread is created (so we have $\mathcal{O}(n^2)$ threads). Each thread modifies only voxels in the direction of the axis orthogonal to the iteration plane, keeping the time complexity the same as with the scatter approach. We further implemented several optimizations and autotuned them via our autotuning framework called Kernel Tuning Toolkit (see Section 4.2). The resulting application is an order of magnitude faster than parallel CPU implementation. Furthermore, it is 2.14× to 5.96× faster than the same algorithm based on a state-of-the-art scatter memory pattern.

Chapter 4

Automatic Optimization

Improving the speed of concrete scientific codes is of high importance. However, it is a task that does not necessarily be done for each new problem from scratch: in some cases, we can identify common code optimization techniques, which can be applied to multiple problems. In such a case, we often can (at least partially) automatize the optimization process. In this chapter, we describe our contribution to two automatic methods: *kernel fusion*, which is a method optimizing multiple kernels (parallel functions in, e. g., CUDA or OpenCL) by connecting them into one, and *autotuning*, which is automatic selection of parameters maximizing application's performance under given conditions (used hardware, program input).

4.1 Kernel Fusion

It is natural to create multiple kernels performing different functions, similar to libraries of multiple functions in commonly used programming languages, such as C. However, partitioning of the code into too many kernels may lead to performance degradation:

- each kernel is compiled and optimized separately; thus, no inlining or cross-kernel optimizations are possible;
- a kernel is executed in a parallel environment, which needs to be initialized and managed; thus, kernel invocation and execution introduces some overhead.

Kernel fusion is an optimization method, which glues together multiple kernel codes into a single kernel and modifies the host code which manages the kernels accordingly (configuring and executing fused kernels). It can improve performance by decreasing kernel execution overhead, improving instruction efficiency, and, probably most importantly, data locality.

With generative programming, code is written in some high-level language or by using patterns, and the OpenCL or CUDA code is then automatically generated. Kernel fusion has a slightly different meaning here – instead of creating fused kernels from unfused ones (explicitly written in C for CUDA or OpenCL), fusion is performed on high-level functions, which the kernel code is generated from. This kind of kernel fusion is enabled

in some frameworks working with algorithmic skeletons [48, 92] and in languages for array programming [17, 57]. In these approaches, fusion is restricted to the scalar function executed by the predefined parallel skeleton or applied to array elements.

Automatic kernel fusion has also been explored in a framework for domain-specific languages [16]. A compiler framework allowing kernel fusion in the domain of relational algebra operations has been introduced in [117].

Performance projection frameworks are used to analyze a given code, predict its performance, and provide optimization hints for the programmer. Kernel fusion is one of the optimization methods considered in [64, 65, 113]. Nevertheless, implementation of the fusion is left to the programmers.

Fusion generators are frameworks, which take OpenCL or CUDA kernels and some programmer annotations as an input and generate fused kernels as an output. The performance projection framework presented in [113] has been extended so that it can analyze user-written kernels and generate fused kernels [114] in the context of CUDA stencil kernels.

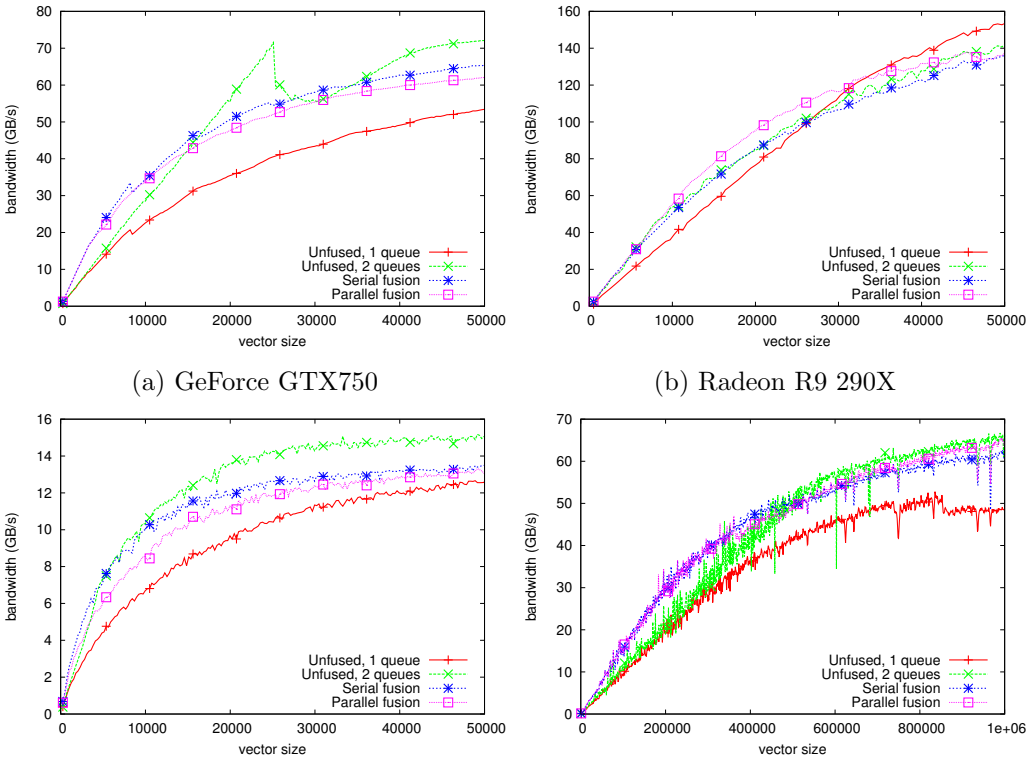
The importance of kernel fusion of BLAS routines has been demonstrated on Krylov subspace methods [6, 23, 90, 102] and equations assembly in finite element methods [35, 49]. In [102], kernel fusion is studied in deeper detail, but only for data-independent kernels. Thus, performance improvements are reported mainly for relatively small inputs. In [90], fusion is employed for both data-dependent and data-independent BLAS kernels.

4.1.1 Contribution

In [30], we studied three types of kernel fusion, applicable to both data-independent and data-dependent kernels, on three platforms programmable with OpenCL: GPU, Xeon Phi, and CPU. We analyzed the effect of kernel fusion on those hardware platforms and showed how it can be used to improve performance.

For data-independent kernels, we propose a *serial fusion* (performing kernels one after another) and *parallel fusion* (executing a code of multiple kernels) at once. On an example of data-independent vector addition (performing $a = b + c$ and $d = e + f$), we show that for small vectors, it is better to fuse the kernels instead of executing them independently (see Figure 4.1), especially on GPUs and Xeon Phi.

Fusion of data-dependent kernels is a non-trivial task since OpenCL and CUDA rely on an explicitly managed memory model and it is not possible to freely transfer data between thread blocks (work-groups) in a single kernel instance. We proposed a basic code transformation method for fusing kernels without cross-thread block data dependency, called *basic data fusion*. The method basically "glues" the kernel codes together and renames conflicting variables. Although no memory transfers are explicitly spared (by moving data into some kind of fastest memory), the fused kernel maintains cache locality better (as the memory footprint of the thread block is limited) or allows the compiler to bypass some redundant memory accesses. The advantage of the basic fusion method is that it requires only very simple source code transformations; thus, it is applicable to a wide area of kernels and has good potential to be automatized. On an example of fusing data-dependent vector addition (performing $a = b + c + d$ by two calls of a kernel adding



(a) GeForce GTX750

(b) Radeon R9 290X

(c) Core i7-3820

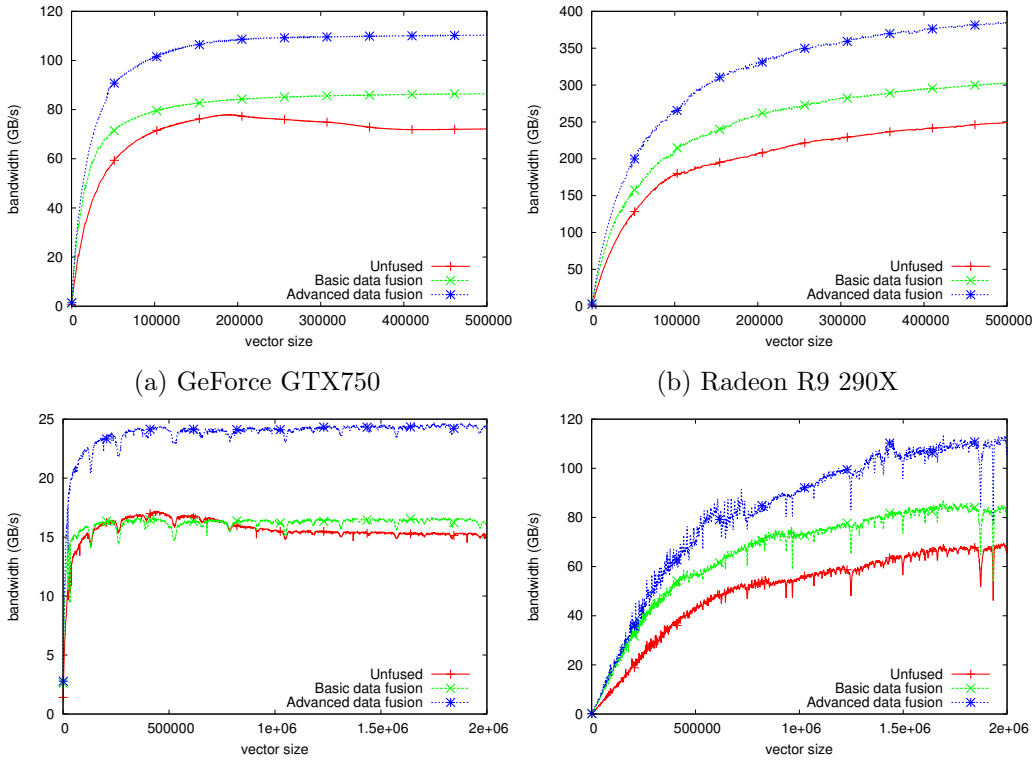
(d) Xeon Phi 5110P

Figure 4.1: Performance of data independent vector additions.

two vectors), we show that the efficiency of the basic transformation method is limited but outperforms unfused kernels (see Figure 4.2).

We address more advanced fusion of data-dependent kernels in [33]. As it is difficult to fuse generic kernels automatically, we limited the type of operations performed by kernels, which can be fused to those performing *map*, *reduce*, or their nested combination. Although some related work allows to fuse map or reduce kernels [17, 57, 92, 103], the first-order function is always serial in these cases. The important feature of our approach is that the first-order function executed by map or reduce can be parallel. Consequently, our approach allows user code to operate on matrix tiles or subvectors instead of matrix or vector scalar elements. It allows us to naturally implement inter-thread optimizations such as tiling (collective reading of shared data into shared memory and reusing them by other threads). The performance benefit of this fusion method is the explicit storing of common data in a faster memory (shared memory or registers), instead of transferring them via global memory in unfused kernels.

We introduced a source-to-source compiler performing the fusion of data-dependent CUDA kernels. The compiler works with a *library of elementary functions* and a *script*, calling functions from the library. It fuses selected functions to improve their performance and preserve the semantics defined by the script. We note that fusing all kernels does not



(a) GeForce GTX750

(b) Radeon R9 290X

(c) Core i7-3820

(d) Xeon Phi 5110P

Figure 4.2: Performance of data dependent vector addition.

necessarily lead to performance improvement in all cases. Thus, the compiler searches and prunes the optimization space to find efficient fusions. It has to decide which kernels to fuse, and how to implement the fusion (e. g., where to place the data).

To demonstrate the performance benefit of kernel fusions generated by our compiler, we have accelerated several sequences of BLAS (Basic Linear Algebra Subprograms) routine calls and compared the result of our compiler to a vendor cuBLAS library. The sequences are described in Table 4.1. The performance improvement over CUBLAS is shown in Table 4.2.

Alongside papers included in this collection, we have also demonstrated kernel fusion effect on element subroutines in FEM (see Section 2.2) [35] and their automation [31]. The simpler automatic fusion usable for kernels implementing parallel *map* was described in [39].

4.2 Autotuning

In recent years, the heterogeneity of both commodity and supercomputer hardware has increased sharply. Accelerators, such as GPU co-processors, are often key to improving the speed and energy efficiency of highly parallel codes. Although languages such

Sequence	Operation
AXPYDOT	$z \leftarrow w - \alpha v$
	$r \leftarrow z^T u$
BiCGK	$q \leftarrow Ap$
	$s \leftarrow A^T r$
GEMVER	$B \leftarrow A + u_1 v_1^T + u_2 v_2^T$
	$x \leftarrow \beta B^T y + z$
	$w \leftarrow \alpha B x$
GESUMMV	$y \leftarrow \alpha A x + \beta B x$
VADD	$x \leftarrow w + y + z$
WAXPBY	$w \leftarrow \alpha x + \beta y$

Table 4.1: Sequences of kernels which can benefit from fusion, adopted from [12].

	Tesla M2090		Tesla K20		GeForce GTX980	
	Perf.	Speedup	Perf.	Speedup	Perf.	Speedup
AXPYDOT	28.38	**1.85**×	30.46	**1.84**×	40.44	**1.67**×
BiCGK	86.97	**2.02**×	79.43	**1.68**×	169.1	**1.99**×
GEMVER	38.27	**1.72**×	38.83	**1.56**×	78.28	**2.24**×
GESUMMV	50.24	**1.33**×	53.49	**1.11**×	86.6	**0.99**×
VADD	15.44	**2.04**×	17.68	**2.14**×	21.07	**1.48**×
WAXPBY	31.88	**2.05**×	33.75	**2.00**×	41.22	**1.93**×

Table 4.2: Performance (in GFlops) of generated sequences and the speedup over cuBLAS using GPUs of three different architectures. Single precision arithmetics is used in all sequences.

as OpenCL allow writing kernels that can be compiled for multiple different processor architectures, performance portability is not ensured. More precisely, the hardware characteristics heavily influence the performance of a given kernel, so its code needs to be adapted for each hardware architecture to achieve optimal performance [56, 71, 82]. Furthermore, kernels' performance is also sensitive to input size, structure, or application settings, so a code optimized for certain input characteristics may run sub-optimally when that change [42, 70, 99].

Addressing the challenges associated with performance optimization and performance portability, *autotuning* has gained a lot of interest. Autotuning of performance-relevant source-code parameters allows to automatically tune applications without hard coding optimizations and thus helps with keeping the performance portable. More precisely, it enables the automatic source code modification according to user-defined *tuning parameters* (properties of the source code, which affects its efficiency) and empirically benchmarks each *tuning configuration* (a unique combination of tuning parameters' values). During this process, it searches for the configuration (and hence a source code), which performs best according to provided input and hardware.

Autotuning can be performed *offline*[1], i. e., before the execution of a tuned code. Offline tuning is easier to implement but does not allow an application to re-tune when its environment changes. *Online* autotuning allows the application to tune itself during runtime by means of changing some runtime parameters. With *dynamic* autotuning, the application can even build the space of different variants during runtime, i. e., it is able to compile tuned kernels during the tuning process. Although several code parameters autotuning frameworks for heterogeneous computing have been introduced [5, 71, 88, 116], they are intended to be used in a *standalone tuning tool*, supporting offline autotuning only. Tighter integration into applications has been recently identified as one of the main challenges in autotuning [7].

For some specific cases, it is meaningful to not only autotune the application code itself but also the application's input. The Fourier Transform is an important tool in many fields of science. Digital signal processing [18], optics [43], astronomy [89], all these are using its variant, so-called Fast Fourier Transform (FFT) to process the data somehow. The size of the signal and parameters of FFT affect the execution time and memory requirements necessary to perform the transformation heavily. In many applications, it is possible to relax the requirements of an 'exact' size of the signal. It might be acceptable to crop (both in the time/frequency domain) or pad it with zeros (in the time domain) prior to the transformation. Therefore, by autotuning, it is possible to search for an optimal signal size and perform the FFT faster. Interestingly, this opportunity is completely missed in the literature: there are a number of other articles that focus on better / faster implementation of FFT for e. g. FPGA [50], autotuning for a specific dimension of the input [72], using autotuning with OpenCL [62] or FFT for specific purposes [25]. However, none of the papers cited above analyses the behavior of the cuFFT in detail or searches for the most efficient configuration of FFT under user-defined constraints.

One of the key features of autotuning frameworks is *searching* the *tuning space* (com-

[1] We adopt the nomenclature from [7].

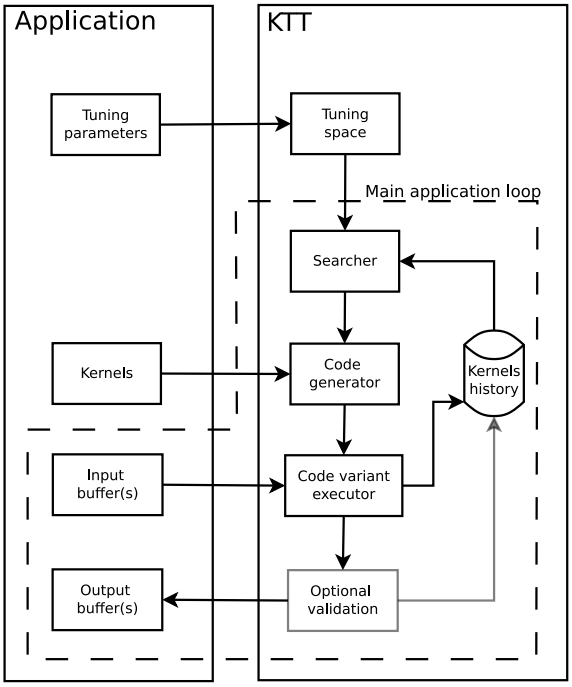


Figure 4.3: Schematic view of KTT architecture. The dashed line shows components that are typically active during dynamic tuning inside the main application loop.

bination of all possible tuning parameters) for a tuning configuration that minimizes a *tuning objective* (usually runtime or power consumption). Autotuning spaces have several properties that make their efficient search difficult. These discrete optimization spaces with many dimensions are known to be non-convex, non-linear, and with poor locality [8]. The time needed to perform a tuning space search can limit the practical usage of autotuning. This happens especially in two cases: (i) when the tuning space is vast, and most of its configurations perform poorly, thus the search takes long; (ii) when the performance depends significantly on input characteristics (e. g., size), which often change, thus the search needs to happen often. The state-of-the-art methods for searching tuning spaces are based on mathematical optimization [8, 116], or they use a surrogate performance/power model built from a sample of the tuning space [27, 28, 52, 86]. Because the relation between tuning parameters and the tuning objective differs with hardware and input, those methods require the autotuning to be repeated from scratch when hardware or input changes.

4.2.1 Contribution

We presented our approach to autotuning in [37, 82], introducing our framework Kernel Tuning Toolkit (KTT). It was designed to simplify the process of autotuning integration by two key features: inter-kernel optimizations and dynamic autotuning. KTT creates an *intermediate layer* between the application and OpenCL or CUDA API, so the ap-

Benchmark	2080Ti	1070	750	K20	Vega56	E5-2650	5110P
BiCG	88.3%	84.7%	81.7%	50.4%	75.6%	46.0%	6.45%
Coulomb 3D	91.8%	91.4%	84.3%	43.2%	65.3%	74.2%	22.2%
GEMM	79.8%	80.6%	91.1%	51.3%	96.3%	37.5%	19.7%
GEMM batched	86.8%	81.4%	90.0%	49.6%	86.0%	27.7%	20.9%
Transpose	87.1%	80.2%	86.3%	64.2%	86.1%	62.5%	10.0%
N-body	89.7%	86.6%	87.7%	40.6%	82.2%	77.7%	29.9%
Reduction	68.7%	87.5%	89.4%	64.1%	71.6%	33.9%	10.1%

Table 4.3: Performance of benchmarks autotuned for various hardware devices (GeForce RTX 2080Ti, GeForce GTX 1070, GeForce GTX 750, Tesla K20, Radeon Vega56, 2x Xeon E5-2650, Xeon Phi 5110P). The performance relative to the theoretical peak of devices is shown.

	best runtime	dynamic tuning
GeForce RTX 2080Ti	1m40s	88% ± 3%
GeForce GTX 1070	5m49s	96% ± 2%
GeForce GTX 750	16m59s	92% ± 4%
GeForce GTX 680	15m12s	94% ± 2%

Table 4.4: The relative performance of dynamically-tuned 3D Fourier reconstruction compared to ōrāculum. The best runtime is measured with ōrāculum.

plication configures and executes kernels indirectly via KTT (its architecture is sketched in Figure 4.3). Therefore, the application source code has to be adapted to incorporate KTT calls. However, once integrated, the application can transparently switch between execution and tuning of the kernels, allowing dynamic autotuning. For example, the application can re-tune itself if it is executed on new hardware or start its execution with already optimized tuning parameters and automatically start re-tuning during runtime when the input changes.

Using KTT, we have developed a *benchmark set* comprising ten autotuned codes. We have executed the benchmark set on multiple hardware devices, including GPUs from NVIDIA and AMD, CPU, and the Xeon Phi. We prove that our autotuned implementations are highly efficient – using just a single tunable code base, they often reach performance close to the theoretical peak – see Table 4.3.

Finally, we proved the applicability of dynamic tuning by KTT in a real application: 3D Fourier Reconstruction in Xmipp [99] (see Section 3.2). The efficiency of 3D Fourier Reconstruction greatly drops when it is tuned for a different GPU or input image size than is executed with: the performance can drop more than 3× when 'bad' input or GPU is used. Therefore, dynamic autotuning can bring considerable speedup. We prepared an experiment using a real-world setup, performing reconstruction of the Brome Mosaic Virus [115] (EMPIAR entry 10010), processing 1,826,160 samples in resolution 156 × 156. In our experiment, the tuning is performed at the beginning of the computation, when both the used hardware and sample size are known. The performance of the dynamically tuned code is compared to the performance of code with ōrāculum (i. e., when the optimal tuning

configuration obtained by the offline tuning using exhaustive search is always known at the beginning of the computation for any hardware and input). We let KTT perform 50 search steps with random search and then continue with the fastest kernel explored. As the random search was used, the experiment was repeated 100 times. Results of this experiment are shown in Table 4.4. As we can see, the performance penalty of dynamic tuning is smaller than the performance penalty we get for a code that was tuned offline for a different hardware device or input size (recall it can drop to less than 33%). This experiment shows that even if the reconstruction program runs in minutes only, dynamic tuning is able to reach better performance than offline tuning. Note that with a longer runtime, dynamic autotuning converges to ōrāculum, provided the optimal configuration is found during the search process.

Besides autotuning source code parameters, we have also targeted a different view of autotuning: we optimize the speed of FFT calculation by changing its input size. If the application allows to change the size of a signal slightly, it can result in much faster Fourier transform computation. We target a popular implementation of FFT for GPU accelerators: the cuFFT library. We analyze the behavior and the performance of the cuFFT library with respect to input sizes and plan settings. We developed a new tool, *cuFFTAdvisor*, which proposes and, by means of autotuning, finds the best configuration of the library for given constraints of input size and plan settings [98]. We experimentally show that our tool is able to propose different settings of the transformation, resulting in an average 6× speedup using fast heuristics and 6.9× speedup using autotuning. We have also demonstrated its usability in the FlexAlign tool (see Section 3.2).

Besides autotuning frameworks, we have also targeted the problem of tuning space search. The tuning space can be large and difficult to search, so the cost of autotuning can outweigh its benefits, especially with dynamic tuning (i. e., the time invested into the autotuning process is longer than time spared by finding more efficient implementation). Therefore, estimating the tuning time in advance is very important in dynamic tuning applications. In [74], we show that it is possible to estimate how many searching steps are needed to achieve reasonable performance. We demonstrated that the portion of tuning space composed of well-performing configurations remains stable for a given problem across different hardware for a majority of cases, even in cases when the well-performing configurations differ. Therefore, it is possible to use historical data to reliably predict the number of tuning steps that are necessary to find a well-performing configuration. We evaluated our hypotheses on a number of HPC benchmarks written in CUDA and OpenCL, using several different generations of GPUs and CPUs. Although we still cannot predict tuning time precisely, as it also depends on the runtime of empirically tested kernels, compilation time, and expected number of kernel's execution, our results presented in [74] confirm our initial hypothesis that histogram of code performance for a tuning space is similar on different hardware, which allows us to use static historical data to make tuning decisions on new hardware when using searching for well-performing configurations.

As we already mentioned, tuning space search is often essential for the efficiency of autotuning. To the best of our knowledge, all state-of-the-art tuning space searchers (see, e. g., [8, 27, 28, 52, 86, 116]) relate tuning parameters to tuning objective (typically the code performance). Because the function giving the performance depends not only

on tuning parameters but also on input and hardware, tuning has to be repeated when hardware or input changes. The essential contribution of our research presented in [32] is the introduction of a novel tuning space search method that breaks the aforementioned relation. Once the tuning space is partially explored for some hardware and input, the *portable model* makes it possible to speed up tuning when input or hardware changes.

Our method mimics the iterative optimization performed by developers. Developers use *profilers* to collect *performance counters* and identify *bottlenecks* (overloaded processor subsystems) on multiple levels of hardware hierarchy. They also understand how the properties of their code (i. e., tuning parameters) are related to performance counters. They iteratively detect bottlenecks and modify the code to reduce stress on the bottlenecks until they reach sufficient code performance. Our method aims to do the same. It requires a training phase, wherein the model is trained to capture how tuning parameters influence performance counters (machine-learning-based analogy to a developer's understanding of the relationship). During autotuning, the method iteratively profiles the code and acquires performance counters analyzes bottlenecks, and determines which performance counters should be changed to soften the bottlenecks using an expert system (analogy to a developer's work with a profiling tool). Then, it uses the previously built model to determine which tuning configurations change performance counters in the required way. Finally, it selects the next tuning configuration to profile (analogy to a developer's modification of the code).

The strength of our method is its ability to build a model using a particular GPU and input and use this model to speed up autotuning of a kernel running on a *different GPU* or processing *different input*. This is possible because the method builds the model of relations between tuning parameters and performance counters and searches for the performance gain based on the performance counters instead of relating tuning parameters directly to the performance. Compared to the performance itself, the tuning parameters affect performance counters measuring the number of operations at some subsystem in a more straightforward and stable way[2]. Therefore, we can build a machine learning model, capturing relations between tuning parameters and performance counters measuring a number of operations. During autotuning, the expert system deduces bottlenecks, using performance counters measuring the stress of the processor subsystems, and asks the machine learning model how to modify tuning parameters to decrease operations causing the bottleneck. The architecture of our searcher is sketched in Figure 4.4.

We have compared our searcher to a state-of-the-art searcher in Kernel Tuner [116]. Using five benchmarks, we have shown that our searcher systematically outperforms Kernel Tuner's best searcher Basin Hopping in both number of tuning steps and convergence time – see examples in Figure 4.5 and Figure 4.6. We also show that it outperforms ML-based Starchart [52] in four out of five benchmarks.

[2]For example, we can have a tuning parameter, which removes some redundant floating-point operations at the cost of lowering the parallelism. Although the optimal value of this parameter depends on GPU and input, the effect of decreasing arithmetic operations and amount of threads persist.

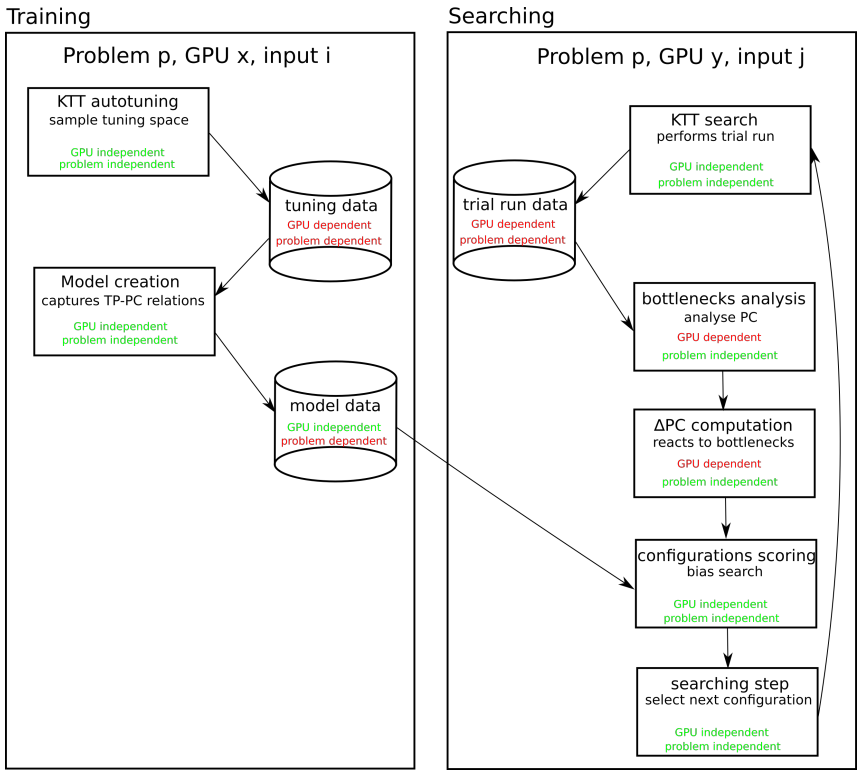


Figure 4.4: Schematic view of the searcher workflow. The boxes show program components and the cylinders show data objects. PC = performance counter, TP = tuning parameter.

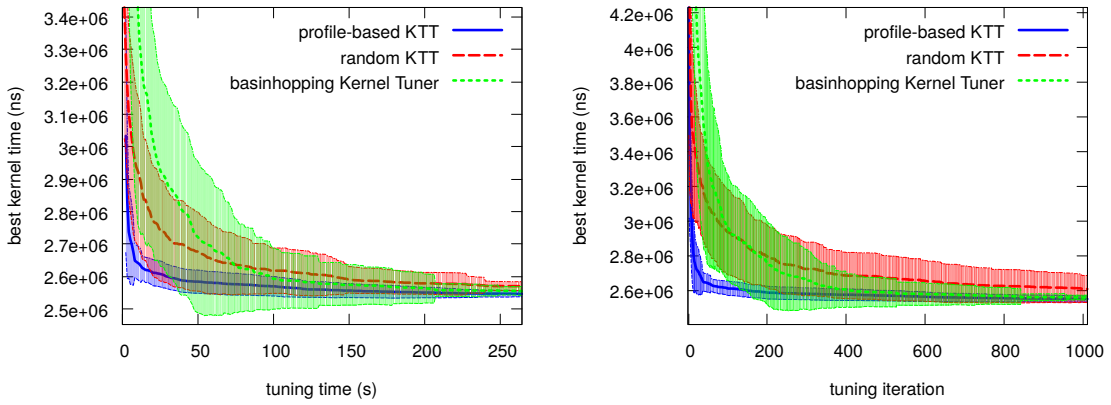


Figure 4.5: Convergence of the GEMM benchmark using KTT and Kernel Tuner. Left: convergence speed in time. Right: comparison of iterations (empirical tests). The solid line shows the average, and the transparent area shows the standard deviation.

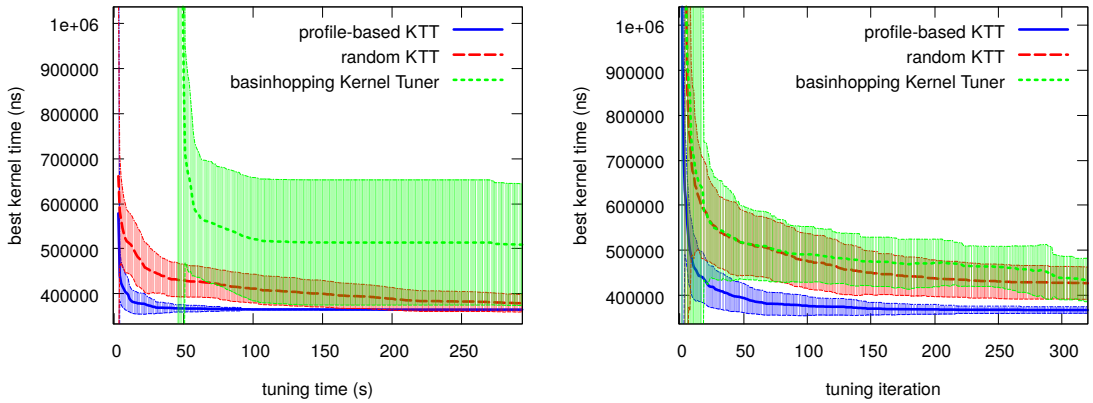


Figure 4.6: Convergence of the Convolution benchmark using KTT and Kernel Tuner. Left: convergence speed in time. Right: comparison of iterations (empirical tests). The solid line shows the average, and the transparent area shows the standard deviation.

Chapter 5

Conclusion

In this text, I have presented a broad area of software performance optimization and my contribution to the state-of-the-art. As the performance can be optimized at multiple levels, it is important to use a holistic approach, spanning from changing computing models through code parallelization and optimization to using specialized hardware. Finally, as some steps in the optimization process can be automatized, I present several ways to do so. This collection includes 14 papers proving my contribution in all aforementioned areas.

In future work, I plan to focus further on software adaptability and its application in scientific software. The profile-based searcher we proposed in [32] opens a wide area of further research. I believe it is possible to replace the expert system in the searcher with a machine learning model, improving the results of the searcher and (with explainable AI) deepening our knowledge of GPU hardware. Furthermore, the performance counters also allow for analyzing and optimizing the tuning space. I plan to further work on the prediction of tuning cost, which would improve the practical applicability of dynamic auto-tuning. I also plan to focus on adaptability in high-level programming, such as algorithmic skeletons, which would enable domain experts to write highly efficient code. Last but not least, I plan to continue accelerating concrete scientific software tools, such as CaverDock and Xmipp. With CaverDock, we are currently working on a version allowing limited flexibility of the receptor, which increases the applicability of CaverDock for cases when tunnel geometry is dynamic (and therefore, current CaverDock may observe bottlenecks that appear and disappear in a real system). Regarding Xmipp, we continuously improve the speed of newly developed algorithms.

Bibliography

[1] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19 – 25, 2015.

[2] V. Abrishami, J. R. Bilbao-Castro, J. Vargas, R. Marabini, J. M. Carazo, and C. O. S. Sorzano. A fast iterative convolution weighting approach for gridding-based direct fourier three-dimensional reconstruction with correction for the contrast transfer function. *Ultramicroscopy*, 157:79 – 87, 2015.

[3] V. Abrishami, J. Vargas, X. Li, Y. Cheng, R. Marabini, C. O. S. Sorzano, and J. M. Carazo. Alignment of direct detection device micrographs using a robust optical flow approach. *Journal of structural biology*, 189(3):163–176, 2015.

[4] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180(1), 2009.

[5] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe. OpenTuner: An extensible framework for program autotuning. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, PACT '14, pages 303–316, 2014.

[6] Hartwig Anzt, Stanimire Tomov, Piotr Luszczek, William Sawyer, and Jack Dongarra. Acceleration of gpu-based krylov solvers via data transfer reduction. *The International Journal of High Performance Computing Applications*, 29(3):366–383, 2015.

[7] P. Balaprakash, J. Dongarra, T. Gamblin, M. Hall, J. K. Hollingsworth, B. Norris, and R. Vuduc. Autotuning in high-performance computing applications. *Proceedings of the IEEE*, 106(11):2068–2083, Nov 2018.

[8] P. Balaprakash, S. M. Wild, and P. D." Hovland. Can search algorithms save large-scale automatic performance tuning? *Procedia Computer Science*, 4:2136 – 2145, 2011.

[9] J. Barbič and D. L. James. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transation on Haptics*, 1(1):39–52, 2008.

[10] C. Basdogan, M. Sedef, M. Harders, and S. Wesarg. Virtual reality supported simulators for training in minimally invasive surgery (invited paper). *IEEE Computer Graphics and Applications*, 27(2):54–66, 2007.

[11] C. Basdogan and M. A. Srinivasan. *Haptic Rendering In Virtual Environments*, pages 157–174. Lawrence Erlbaum Associates, 2001.

[12] G. Belter, E. R. Jessup, I. Karlin, and J. G. Siek. Automating the generation of composed linear algebra kernels. In *Proceedings of the Conference on High Performance Computing, Networking, Storage and Analysis (SC09)*, pages 1–12. ACM, 2009.

[13] D. Branduardi, F. L. Gervasio, and M. Parrinello. From A to B in free energy space. *The Journal of Chemical Physics*, 126(5):054103, 2007.

[14] J. Brezovsky, P. Babkova, O. Degtjarik, A. Fortova, A. Gora, I. Iermak, P. Rezacova, P. Dvorak, I. Kuta Smatanova, Z. Prokop, R. Chaloupkova, and J. Damborsky. Engineering a de novo transport tunnel. *ACS Catalysis*, 6(11):7597–7610, 2016.

[15] M. Bro-Nielsen. Finite element modeling in surgery simulation. In *Proceedings of the IEEE*, pages 490–503, 1998.

[16] K.J. Brown, A.K. Sujeeth, Hyouk Joong Lee, T. Rompf, H. Chafi, M. Odersky, and K. Olukotun. A heterogeneous parallel framework for domain-specific languages. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, 2011.

[17] B. Catanzaro, M. Garland, and K. Keutzer. Copperhead: compiling an embedded data parallel language. In *The 16th ACM Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2011.

[18] Chi-Tsong Chen. *Digital signal processing: spectral computation and filter design.* Oxford University Press, Inc., 2000.

[19] E. Chovancová, A. Pavelka, P. Beneš, O. Strnad, J. Brezovský, B. Kozlíková, and et al. CAVER 3.0: A tool for the analysis of transport pathways in dynamic protein structures. *PLoS Computational Biology*, 8(10), 2012.

[20] S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions On Visualization and Computer Graphics*, 5(1):62–73, 1999.

[21] R. A. Crowther, D. J. DeRosier, and F. R. S. Klug. The reconstruction of a three-dimensional structure from projections and its application to electron microscopy. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 317(1530):319–340, 1970.

[22] J. Damborsky and J. Brezovsky. Computational tools for designing and engineering biocatalysts. *Current Opinion in Chemical Biology*, 13(1):26–34, 2009. Biocatalysis and Biotransformation/Bioinorganic Chemistry.

[23] M. M. Dehnavi, D. M. Fernández, and D. Giannacopoulos. Enhancing the performance of conjugate gradient solvers on graphic processing units. *IEEE Transactions on Magnetics*, 47(5), 2011.

[24] D. Devaurs, L. Bouard, M. Vaisset, C. Zanon, I. Al-Bluwi, R. Iehl, T. Siméon, and J. Cortés. MoMA-LigPath: a web server to simulate protein–ligand unbinding. *Nucleic Acids Research*, 41(W1), 2013.

[25] Sofia Dimoudi, Karel Adamek, Prabu Thiagaraj, Scott M Ransom, Aris Karastergiou, and Wesley Armour. A gpu implementation of the correlation technique for real-time fourier domain pulsar acceleration searches. *The Astrophysical Journal Supplement Series*, 239(2):28, 2018.

[26] T. J. A. Ewing, S. Makino, A. G. Skillman, and I. D. Kuntz. DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer-Aided Molecular Design*, 15(5), 2001.

[27] T. L. Falch and A. C. Elster. Machine learning based auto-tuning for enhanced OpenCL performance portability. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 2015.

[28] W. Feng and T. S. Abdelrahman. A sampling based strategy to automatic performance tuning of gpu programs. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017.

[29] J. Filipovic, O. Vavra, J. Plhak, D. Bednar, S. M. Marques, J. Brezovsky, L. Matyska, and J. Damborsky. Caverdock: A novel method for the fast analysis of ligand transport. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 17(5):1625–1638, 2020.

[30] J. Filipovič and S. Benkner. Opencl kernel fusion for gpu, xeon phi and cpu. In *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 98–105, 2015.

[31] J. Filipovič, J. Fousek., B. Lakomý, and M. Madzin. Automatically optimized GPU acceleration of element subroutines in finite element method. In *Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, 2012.

[32] J. Filipovič, J. Hozzová, Nezarat A., J. Oĺha, and F. Petrovič. Using hardware performance counters to speed up autotuning convergence on gpus. *Journal of Parallel and Distributed Computing*, 160:16–35, 2022.

[33] J. Filipovič, M. Madzin, J. Fousek, and L. Matyska. Optimizing CUDA code by kernel fusion: application on BLAS. *Journal of Supercomputing*, 71(10):3934–3957, 2015.

[34] J. Filipovič, J. Pazúriková, A. Křenek, and V. Spiwok. Accelerated rmsd calculation for molecular metadynamics. In *Proceedings of the 2016 European Simulation and Modelling Conference*, pages 278–280, 2016.

[35] J. Filipovič, I. Peterlík, and J. Fousek. GPU acceleration of equations assembly in finite elements method – preliminary results. In *Symposium on Application Accelerators in High-Performance Computing (SAAHPC)*, 2009.

[36] J. Filipovič, I. Peterlík, and L. Matyska. On-line precomputation algorithm for real-time haptic interaction with non-linear deformable bodies. In *Third Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC)*, 2009.

[37] J. Filipovič, F. Petrovič, and S. Benkner. Autotuning of OpenCL kernels with global optimizations. In *Proceedings of the 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy Efficient HPC Systems (ANDARE '17)*, 2017.

[38] J. Filipovič, J Plhák, and D. Střelák. Acceleration of drmsd calculation and efficient usage of gpu caches. In *2015 International Conference on High Performance Computing Simulation (HPCS)*, 2015.

[39] J. Fousek, J. Filipovič, and M. Madzin. Automatic fusions of CUDA-GPU kernels for parallel map. In *Second International workshop on highly-efficient accelerators and reconfigurable technologies (HEART)*, pages 42–47, 2011.

[40] J. Fousek, T. Golembiovský, J. Filipovič, and I. Peterlík. Haptic rendering based on rbf approximation from dynamically updated data. In *Sixth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'10)–Selected Papers*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[41] R. M. Glaeser. Chapter two - specimen behavior in the electron beam. In R. A. Crowther, editor, *The Resolution Revolution: Recent Advances In cryoEM*, volume 579 of *Methods in Enzymology*, pages 19 – 50. Academic Press, 2016.

[42] S. G. D. Gonzalo, S. D. Hammond, C. R. Trott, and W. M. a. Hwu. Revisiting online autotuning for sparse-matrix vector multiplication kernels on next-generation architectures. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2017.

[43] Joseph W Goodman and P Sutton. Introduction to fourier optics. *Quantum and Semiclassical Optics-Journal of the European Optical Society Part B*, 8(5):1095, 1996.

[44] Richard Henderson. Overview and future of single particle electron cryomicroscopy. *Arch Biochem Biophys*, 581:19–24, Sep 2015.

[45] D. Herreros, J. Kiska, E. Ramírez-Aportela, J. Filipovic, J.M. Carazo, and C.O.S. Sorzano. Zart: A novel multiresolution reconstruction algorithm with motion-blur correction for single particle analysis. *Journal of Molecular Biology*, 435(9):168088, 2023.

[46] D. Herreros, R. R. Lederman, J. Krieger, A. Jiménez-Moreno, M. Martínez, D. Myška, D. Střelák, J. Filipovič, I. Bahar, J. M. Carazo, and C. O. S. Sorzano. Approximating deformation fields for the analysis of continuous heterogeneity of biological macromolecules by 3d zernike polynomials. *IUCrJ*, 8(6), 2021.

[47] D. Herreros, R. R. Lederman, J. M. Krieger, A. Jiménez-Moreno, M. Martínez, D. Myška, D. Strelak, J. Filipovic, C.O.S. Sorzano, and J.M. Carazo. Estimating conformational landscapes from cryo-em particles by 3d zernike polynomials. *Nature Communications*, 14(1):154, 2023.

[48] J. Hoberock and N. Bell. Thrust: A parallel template library, 2009.

[49] X. S. Hu, G. Hsieh, L. Tang, S. D. Hammond, D. Z. Chen, M. Niemier, and R. F. Barrett. GPU acceleration of data assembly in finite element methods and its energy implications. In *Proceedings of the 2013 IEEE 24th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2013.

[50] Muhammad Ibrahim and Omar Khan. Performance analysis of fast fourier transform on field programmable gate arrays and graphic cards. In *Computing, Electronic and Electrical Engineering (ICE Cube), 2016 International Conference on*, pages 158–162. IEEE, 2016.

[51] M. Jamroz and A. Kolinski. ClusCo: clustering and comparison of protein models. *BMC Bioinformatics*, 14(62), 2013.

[52] W. Jia, K. A. Shaw, and M. Martonosi. Starchart: Hardware and software optimization using recursive partitioning regression trees. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, 2013.

[53] A. Jiménez-Moreno, Střelák D., J. Filipovič, J. M. Carazo, and C. O. S. Sorzano. Deepalign, a 3d alignment method based on regionalized deep learning for cryo-em. *Journal of Structural Biology*, 213(2), 2021.

[54] S. Jonic, C. O. S. Sorzano, P. Thévenaz, C. El-Bez, S. De Carlo, and M. Unser. Spline-based image-to-volume registration for three-dimensional electron microscopy. *Ultramicroscopy*, 103/104:303–317, 2005.

[55] D. Kimanius, B. O. Forsberg, S.H.W. Scheres, and E. Lindahl. Accelerated cryo-em structure determination with parallelisation using gpus in relion-2. *eLife*, 5:e18722, 2016.

[56] J. Kurzak, S. Tomov, and J. Dongarra. Autotuning GEMM kernels for the Fermi GPU. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2045–2057, 2012.

[57] B. Larsen. Simple optimizations for an applicative array language for graphics processors. In *Proceedings of the sixth workshop on Declarative aspects of multicore programming (DAMP)*, 2011.

[58] S. D. Laycock and A. M. Day. A survey of haptic rendering techniques. In *Computer graphics forum*, volume 26, pages 50–65. Wiley Online Library, 2007.

[59] P. H. Lee, K. L. Kuo, P.Y. Chu, E. M. Liu, and J. H. Lin. SLITHER: a web server for generating contiguous conformations of substrate molecules entering into deep active sites of proteins or migrating through channels in membrane transporters. *Nucleic Acid Research*, 37(W559-64), 2009.

[60] X. Li, N. Grigorieff, and Y. Cheng. Gpu-enabled frealign: Accelerating single particle 3d reconstruction and refinement in fourier space on graphics processors. *Journal of Structural Biology*, 172(3):407 – 412, 2010.

[61] Y. Li, Y.-Q. Zhang, Y.-Q. Liu, G.-P. Long, and H.-P. Jia. MPFFT: An auto-tuning FFT library for OpenCL GPUs. *Journal of Computer Science and Technology*, 28(1):90–105, 2013.

[62] Yan Li, Yunquan Zhang, Haipeng Jia, Guoping Long, and Ke Wang. Automatic fft performance tuning on opencl gpus. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 228–235. IEEE, 2011.

[63] S.M. Marques, L. Daniel, T. Buryska, Z. Prokop, J. Brezovsky, and J. Damborsky. Enzyme tunnels and gates as relevant targets in drug design. *Medicinal Research Reviews*, 37(5):1095–1139, 2017.

[64] J. Meng, V. A. Morozov, V. Vishwanath, and K. Kumaran. Dataflow-driven GPU performance projection for multi-kernel transformations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC12)*, 2012.

[65] J. Meng, X. Wu, V. Morozov, V. Vishwanath, K. Kumaran, and V. Taylor. SKOPE: A framework for modeling and exploring workload behavior. In *Proceedings of the 11th ACM Conference on Computing Frontiers (CF)*, 2014.

[66] S. Misra, A. M. Okamura, and K. T. Ramesh. Force feedback is noticeably different for linear versus nonlinear elastic tissue models. In *Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2007.

[67] T. Mohammad, Y. Mathur, and M. I. Hassan. InstaDock: A single-click graphical user interface for molecular docking-based virtual high-throughput screening. *Briefings in Bioinformatics*, 22(4), 10 2020.

[68] G. M. Morris, R. Huey, W. Lindstrom, M. F. Sanner, R. K. Belew, D. S. Goodsell, and A. J. Olson. AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry*, 30(16), 2009.

[69] P. Němcová, J. Hozzová, and J. Filipovič. Improving ligand transport trajectory within flexible receptor in caverdock. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, pages 619–626, 2022.

[70] C. Nugteren. CLBlast: A tuned OpenCL BLAS library. In *Proceedings of the International Workshop on OpenCL*, IWOCL '18, pages 5:1–5:10. ACM, 2018.

[71] C. Nugteren and V. Codreanu. CLTune: A generic auto-tuner for OpenCL kernels. In *Proceedings of the IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2015.

[72] Akira Nukada and Satoshi Matsuoka. Auto-tuning 3-d fft library for cuda gpus. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 30. ACM, 2009.

[73] NVIDIA. cufft: The cuda fast fourier transform library, 2022.

[74] J. Oľha, J. Hozzová, J. Fousek, and J. Filipovič. Exploiting historical data: Pruning autotuning spaces and estimating the number of tuning steps. *Concurrency and Computation: Practice and Experience*, 32(21), 2020.

[75] M. Olšák, J. Filipovič, and M. Prokop. Fastgrid — the accelerated autogrid potential maps generation for molecular docking. *Computing and Informatics*, 29(6+), 2010.

[76] P. A. Penczek. Chapter one - fundamentals of three-dimensional reconstruction from projections. In *Cryo-EM, Part B: 3-D Reconstruction*, volume 482 of *Methods in Enzymology*, pages 1 – 33. Academic Press, 2010.

[77] J. R. Perilla and K. Schulten. Physical properties of the hiv-1 capsid from all-atom molecular dynamics simulations. *Nature Communications*, 8, 2017.

[78] I. Peterlík. Efficient precomputation of configuration space for haptic deformation modeling. In *Conference on Human System Interactions*, 2008.

[79] I. Peterlík, J. Filipovič, and L. Matyska. Distributed construction of configuration spaces for real-time haptic deformation modeling. *IEEE Transactions on Industrial Electronics*, 58(8), 2011.

[80] I. Peterlík and L. Matyska. An algorithm of state-space precomputation allowing non-linear haptic deformation modelling using finite element method. In *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2007.

[81] F. Petrovič and J. Filipovič. Kernel tuning toolkit. *SoftwareX*, 22:101385, 2023.

[82] F. Petrovič, D. Střelák, J. Hozzová, J. Oľha, R. Trembecký, S. Benkner, and J. Filipovič. A benchmark set of highly-efficient CUDA and OpenCL kernels and its dynamic autotuning with kernel tuning toolkit. *Future Generation Computer Systems*, 108:161–177, 2020.

[83] G. Pinto, O. Vávra, J. Filipovič, J. Štourač, D. Bednář, and J. Damborský. Fast screening of binding and unbinding of inhibitors using novel software tool CaverDock. *Frontiers in Chemistry*, 7, 2019.

[84] G. P. Pinto, O. Vávra, S. M. Marques, J. Filipovič, D. Bednář, and J. Damborský. Screening of world approved drugs against highly dynamical spike glycoprotein of sars-cov-2 using caverdock and machine learning. *Computational and Structural Biotechnology Journal*, 19:3187–3197, 2021.

[85] D. C. Popescu and M. Compton. A model for efficient and accurate interaction with elastic objects in haptic virtual environments. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 245–250, New York, NY, USA, 2003. ACM Press.

[86] J Price and S. McIntosh-Smith. Improving auto-tuning convergence times with dynamically generated predictive performance models. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, 2015.

[87] M. Radermacher. *Weighted Back-Projection Methods*, pages 91–115. Springer US, Boston, MA, 1992.

[88] A. Rasch and S. Gorlatch. ATF: A generic directive-based auto-tuning framework. *Concurrency and Computation: Practice and Experience*, 0(0):e4423, 2018.

[89] Daniel N Rockmore. The fft: an algorithm the whole family can use. *Computing in Science & Engineering*, 2(1):60–64, 2000.

[90] K. Rupp, J. Weinbub, A. Jüngel, and T Grasser. Pipelined iterative solvers with kernel fusion for graphics processing units. *PAMM*, 14(1), 2014.

[91] R. Salomon-Ferrer, D. A. Case, and R. C. Walker. An overview of the Amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2013.

[92] S. Sato and H. Iwasaki. A skeletal parallel framework with fusion optimizer for GPGPU programming. In *Programming Languages and Systems*, volume 5904 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009.

[93] D. Sehnal, R. Svobodová Vařeková, K. Berka, L. Pravda, V. Navrátilová, P. Banáš, C. M. Ionescu, M. Otyepka, and J. Koča. MOLE 2.0: advanced approach for analysis of biomacromolecular channels. *Journal of Cheminformatics*, 39(5), 2013.

[94] C. O. S. Sorzano, J. Vargas, J. Otón, J. L. Vilas, M. Kazemi, R. Melero, L. del Caño, J. Cuenca, P. Conesa, J. Gómez-Blanco, R. Marabini, and J. M. Carazo. A survey of the use of iterative reconstruction algorithms in electron microscopy. *BioMed Research Intl.*, 2017:6482567, 2017.

[95] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry*, 28(16), 2007.

[96] D. Střelák, J. Filipovič, A. Jiménez-Moreno, J. M. Carazo, and C. O. S. Sorzano. Flexalign: An accurate and fast algorithm for movie alignment in cryo-electron microscopy. *Electronics*, 9(6):1040, 2020.

[97] D. Střelák, D. Myška, F. Petrovič, J. Polák, J. Ol'ha, and J. Filipovič. Umpalumpa: a framework for efficient execution of complex image processing workloads on heterogeneous nodes. *Computing*, pages 1–29, 2023.

[98] D. Střelák and J. Filipovič. Performance analysis and autotuning setup of the cufft library. In *Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems*, ANDARE '18, New York, NY, USA, 2018. Association for Computing Machinery.

[99] D. Střelák, C. O. S. Sorzano, J. M. Carazo, and J. Filipovič. A GPU acceleration of 3D Fourier reconstruction in Cryo-EM. *The International Journal of High Performance Computing Applications*, 0, 2019.

[100] H. Su, W. Wen, X. Du, X. Lu, M. Liao, and D. Li. Gerelion: Gpu-enhanced parallel implementation of single particle cryo-em image processing. *bioRxiv*, 2016.

[101] M. Suan Li and B. Khanh Mai. Steered molecular dynamics-a promising tool for drug design. *Current Bioinformatics*, 7(4):342–351, 2012.

[102] S. Tabik, Ortegam G., and E. M. Garzón. Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study. *The Journal of Supercomputing*, 70(2), 2014.

[103] D. Tarditi, S. Puri, and J. Oglesby. Accelerator: using data parallelism to program GPUs for general-purpose uses. *SIGARCH Computer Architure News*, 34(5), 2006.

[104] R. Thomsen and M. H. Christensen. MolDock: A new technique for high-accuracy molecular docking. *Journal of Medicinal Chemistry*, 49(11):3315–3321, 2006.

[105] P. Tiwary, V. Limongelli, M. Salvalaglio, and M. Parrinello. Kinetics of protein–ligand unbinding: Predicting pathways, rates, and rate-limiting steps. *Proceedings of the National Academy of Sciences*, 112(5):E386–E391, 2015.

[106] A. E. Torda and W. F. van Gunsteren. Algorithms for clustering molecular dynamics configurations. *Journal of Computational Chemistry*, 15(12), 1994.

[107] J.O. Tørring, B. van Werkhoven, F. Petrovč, F.-J. Willemsen, J. Filipovič, and A.C. Elster. Towards a benchmarking suite for kernel tuners. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 724–733. IEEE, 2023.

[108] O. Trott and A. J. Olson. Autodock vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2), 2010.

[109] O. Vavra, J. Beranek, J. Stourac, M. Surkovsky, J. Filipovic, J. Damborsky, J. Martinovic, and D. Bednar. pycaverdock: Python implementation of the popular tool for analysis of ligand transport with advanced caching and batch calculation support. *Bioinformatics*, 39(8):btad443, 2023.

[110] O. Vavra, J. Filipovic, J. Plhak, D. Bednar, S. M. Marques, J. Brezovsky, J. Stourac, L. Matyska, and J. Damborsky. CaverDock: a molecular docking-based tool to analyse ligand transport through protein tunnels and channels. *Bioinformatics*, 35(23):4986–4993, 05 2019.

[111] V. Volkov and J. W. Demmel. Benchmarking GPUs to tune dense linear algebra. In *ACM/IEEE conference on Supercomputing (SC)*, 2008.

[112] J. Štourač, O. Vávra, P. Kokkonen, J. Filipovič, G. P. Pinto, J. Brezovský, J. Damborský, and D. Bednář. Caver Web 1.0: identification of tunnels and channels in proteins and analysis of ligand transport. *Nucleic Acids Research*, 47(W1):W414–W422, 2019.

[113] M. Wahib and N. Maruyama. Scalable kernel fusion for memory-bound GPU applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC14)*, 2014.

[114] M. Wahib and N. Maruyama. Automated GPU kernel transformations in large-scale production stencil applications. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2015.

[115] Z. Wang, C. F. Hryc, B. Bammes, P. V. Afonine, J. Jakana, D.-H. Chen, X. Liu, M. L. Baker, C. Kao, S. J. Ludtke, M. F. Schmid, P. D. Adams, and W. Chiu. An atomic model of brome mosaic virus using direct electron detection and real-space optimization. *Nat Commun*, 5:4808, 2014.

[116] B. van Werkhoven. Kernel tuner: A search-optimizing gpu code auto-tuner. *Future Generation Computer Systems*, 90:347 – 358, 2019.

[117] H. Wu, G. Diamos, S. Cadambi, and S. Yalamanchili. Kernel weaver: Automatically fusing database primitives for efficient GPU computation. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[118] E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov. MolAxis: Efficient and accurate identification of channels in macromolecules. *Proteins: Structure, Function, and Bioinformatics*, 73(1):72–86, 2008.

[119] X. Zhang, X. Zhang, and Z.H. Zhou. Low cost, high performance gpu computing solution for atomic resolution cryoem single-particle reconstruction. *Journal of Structural Biology*, 172(3):400 – 406, 2010.

[120] S. Q. Zheng, E. Palovcak, J. P. Armache, K. A. Verba, Y. Cheng, and D. A. Agard. Motioncor2: anisotropic correction of beam-induced motion for improved cryo-electron microscopy. *Nature methods*, 14(4):331–332, 2017.

[121] J. Zivanov, T. Nakane, B. O. Forsberg, D. Kimanius, W. J. H. Hagen, E. Lindahl, and S. H. W. Scheres. New tools for automated high-resolution cryo-em structure determination in relion-3. *Elife*, 7, 2018.

Part II

Collection of Articles

Articles are not available in public version of this thesis.