M U N I

Annex No. 10 to the MU Directive on Habilitation Procedures and Professor Appointment Procedures

HABILITATION THESIS REVIEWER'S REPORT

Masaryk University	
Applicant	Jiří Filipovič, Ph.D.
Habilitation thesis	Software Performance Optimization in Scientific Computing
Reviewer	Richard Vuduc, Ph.D.
Reviewer's home unit, institution	School of Computational Science and Engineering, Georgia Institute of Technology (“Georgia Tech”)

This habilitation thesis concerns how to make software for scientific computing efficient ***in practice***. Efficiency “in practice” refers to using as little wall clock time as possible when running the software on a *real* computer system. By contrast, efficiency “in theory” is achieved when a new algorithm or design is shown to use fewer operations or less memory than a baseline algorithm when running on an *abstract* machine with some idealized notion of cost. The present thesis takes advantage of its practical outlook, leading to an immediate impact on real applications, as well as contributing novel methodologies that I estimate will have an impact on future applications.

The thesis describes several distinct contributions, all of which have been published in peer-reviewed journals, conferences, or workshop papers. These contributions are organized into three “parts” (Chapters 2, 3, and 4, respectively).

The first part focuses on two technical contributions that directly improve specific applications that are based on computer simulations (Chapter 2).

In its first contribution, this thesis develops novel strategies to improve the speed of a phenomenon in biomolecular systems known as *molecular docking* (Section 2.1), wherein a small molecule attaches itself to a larger molecule. Molecular docking is of interest in the design of drugs, for instance. This phenomenon is typically studied by physically accurate but expensive, long-running computer-based molecular dynamics simulations. Expertly designed heuristics can reduce this cost, but deriving such heuristics is impractical when one wishes to analyze many configurations (e.g., screening hundreds of drug candidates) since the heuristics may require tuning for each scenario. To address this problem, the thesis proposes a mathematical method to approximate the shapes of the molecules in such a way that it systematically narrows the configurations that must be explored. While I am not an expert in this domain, I do believe the proposed strategy is technically sound. More importantly, since the method is implemented, evaluated on realistic test cases, and made publicly available, independent domain experts will be able to try it easily, which enhances its potential for impact.

The second contribution is a system that enables real-time virtual surgical simulations (Section 2.2). In this setting, a human performs a simulated surgical procedure in a machine that provides haptic feedback that must be calculated in real-time. The calculations involve solving a nonlinear system of partial differential equations. The challenge is that a solution must be computed in response to what the human operator does. To speed up these

calculations, the thesis describes a novel *precomputation* scheme. The proposed technique anticipates possible human actions and, using a pool of servers, precomputes solutions, selecting the appropriate one once the human input is received. The precomputation is based on further approximations to ensure the overall system can meet the real-time constraints of the application. Again, while I am not an expert in this area, I find the proposed strategy to be a creative one. The application is also challenging and exciting, not to mention one that can have an immediate and possibly commercial or industrial impact.

The second part (Chapter 3) develops methods to speed up computer simulations using graphics co-processors, or GPUs. Compared to conventional multicore CPU processors, GPUs provide many more, but simpler, cores. Whereas multicore CPUs are best suited to concurrent or task-parallel computations where the threads are loosely coupled and the performance of each thread is the performance limiter (due to the presence of irregular branching and control, for instance), GPUs are well-suited to "PRAM-style" computations that have a lot of fine-grained synchronous parallelism (where one can exploit many threads that each thread carries out more or less the same operations as any other). Although GPUs have been around for a long time, modern programmers have little direct exposure to parallel processing, so exploiting GPUs remains a challenging activity, and while we know many principles for how to exploit them, there is still much to explore. As such, it is still common to see case studies that help to identify these principles. This thesis presents two such case studies.

The first case study of GPU acceleration is for computational chemistry via particle simulations, where one critical bottleneck is the calculation of distances between atoms. Such calculations are known to be compute-bound, meaning their speed is limited only by the time to perform arithmetic operations, instead of the time to move data or communicate. As such, one would guess that a simple approach to parallelization, ignoring communication cost, would be sufficient to get good performance. Indeed, the fundamental design of a GPU system is to provide more hardware support for many concurrent threads than for caches or other components of a memory hierarchy, which might otherwise communication costs (as they do on multicore CPUs). The surprising finding of this case study is that exploiting caches (and registers, which function as a kind of cache on GPUs) is critical, contrary to *a priori* expectations. But doing also naïvely incurs overheads, and overcoming those requires combining several other performance-engineering techniques. Overall, the case study is a well-designed and well-written example of the effort required, and I expect it will serve as an important example for future efforts to automate this engineering process.

The second case study involves volumetric image reconstruction from electron microscopy data. Like the previous case study, this one brings multiple performance-engineering techniques together. One strategy is to replace a "write-oriented" algorithm with a "read-oriented" one, which ends up reducing communication costs. Another is to precompute partial results, thereby reducing the overall algorithmic complexity of the calculation. A third is to carefully tune the innermost loops. The tuning challenge is that there are many tuning options and no clear way to distinguish them without doing experiments. But rather than conduct this tuning by hand, the case study develops an automated tuning method. The technical contribution in this case is both identifying the tuning options, which requires expert judgement, as well as efficiently exploring these options. When combined, the resulting implementation is an order of magnitude faster than an already-parallel baseline.

While the case studies are valuable and impactful, they are also specific, raising the natural question of how well the methods can be generalized. The third part of the thesis (Chapter 4) takes up this question directly, proposing generic techniques that can be applied more broadly.

The first proposed technique is kernel fusion, where distinct parallel functions are combined into a single parallel function. The goals of fusion are to reduce the overheads of separate invocations of the parallel functions as well as to improve their data locality. This form of fusion is related to classical loop fusion in the compiler literature except the program representation considered here has two significant differences: it is explicitly parallel and explicitly manages data movement through a multi-level memory hierarchy. Nevertheless, as with classical loop fusion, in kernel fusion it can be hard to reason about when the fusion is legal and when it will pay off, as fusion increases the memory and code footprint of the function. The thesis studies kernel fusion in this setting, showing the same benefits of classical fusion can extend to this other type of program representation. Moreover, it proposes higher-level abstract primitives that can be used to simplify the program representation and, thereby, enable automatic kernel fusion to be applied in more settings. Lastly, to address the issue of when to apply kernel fusion, it develops a methodology to analyze the efficacy of kernel fusion using automated experiments. This combination of techniques appears quite promising, leading, for instance, to implementations of widely used dense linear algebra routines that outperform a strong baseline (a very fast, hand-tuned vendor library), by just over a factor of two.

A common theme of the contributions described above is the use of automated performance-tuning experiments to find the best implementation, an area that has come to be known (since the late 1990s) as autotuning. The thesis contributes a new autotuning framework called the Kernel Tuning Toolkit (KTT). It aims to simplify how end-user developers can make their programs “tunable,” and by exposing the tuning options to the KTT system, automate what would otherwise be a manual tuning process. While several systems exist with similar goals, the KTT is the most comprehensive of these to date, comprising a programming interface for users, a code generator, a search engine (to run automated experiments and explore the tuning space using a variety of heuristic algorithms), and targeting GPUs. KTT is applied to several case studies, including the one related to image reconstruction mentioned previously.

One aspect of the autotuning problem is especially vexing, namely, how to retune an implementation portably. Portability in this case refers specifically to two scenarios, namely, when the input changes and when the hardware changes. The strategy employed in the thesis is to learn models during the tuning process. In particular, the proposed strategy records detailed profiles of the program as it is tuned, collecting performance-counter data observable on most modern processors, and then uses these profiles to construct empirical models of the program’s performance behavior as a function of those data. This model can be used to predict performance as the tuning parameters changed when moving to a new set of inputs or a new platform. This approach is implemented in a tool called the Kernel Tuner, which is then shown to outperform a state-of-the-art baseline tool (Starchart) that uses different machine learning techniques.

Reviewer's questions for the habilitation thesis defence (number of questions up to the reviewer)

All questions I had about the thesis are addressed in the attached publications. I have no additional questions beyond those.

Conclusion

This habilitation thesis contains includes a wide range of contributions, some with immediate practical impact (application-specific performance enhancements, Chapter 2), some concerning an important class of modern machines (performance engineering for GPUs,

Chapter 3), and others that try to use those experiences to derive general lessons into tools that can be applied more broadly (autotuning tools, Chapter 4). Overall, this variety in the styles of work conducted demonstrates the versatility and flexibility of the author's research program, which is nevertheless coherently focused on performance engineering for scientific simulations.

To understand the relative impact of this work, I might compare Filipovič to two highly recognized scholars working in similar areas of performance engineering within the US: Tze Meng Low (Carnegie Mellon) and Richard Veras (U. Oklahoma). Low is a recently tenured Associate Professor who is slightly more senior than Filipovič, and Veras a mid-career Assistant Professor who is more junior. Filipovič's overall citation counts and h-index are roughly the same as Low and Veras, indicating that, objectively speaking, we might expect similar levels of positive impact and career trajectories. From a qualitative or style perspective, I would characterize Low's and Veras's work as more theoretical in style and more focused on more general numerical tools from linear algebra and graph analytics, whereas Filipovič's work is deeper in scientific computing applications than Low's and Veras's as well as more practical given its specific GPU focus. In any case, the quality of Low's work is highly regarded, and I find Filipovič's work to be of comparable quality.

Given its overall breadth and depth and positive comparisons to work by peer scholars, the habilitation entitled "Software performance optimization in scientific computing" by Jiří Filipovič, Ph.D., fulfils requirements expected of a habilitation thesis in the field of Computer Science.

Date: April 8, 2024 Signature: