Performance-Centric System Design

Torsten Hoefler
Swiss Federal Institute of Technology, ETH Zürich

Performance-Centric System Design
- modeling, programming, and architecture -

Torsten Hoefler
ETH Zürich

December 9, 2015

Abstract

We live in the era of ubiquitous computation ranging from billions of mobile devices such as cell phones or smart watches processing data in the cloud to high-performance computing systems simulating next-generation medicine. Such computations are executed in large-scale or "warehouse-scale" data centers and high-performance computers. These crucial massive computers cost multiple hundreds of millions to billions of Francs to construct and tens of millions of Francs each year to operate. Thus, the efficiency of such computers is of utmost importance. My work focuses on performance-centric parallel programming and ranges from the design of efficient parallel computers and datacenter architectures to parallel programming models and algorithms. The key feature in his approach is to model the performance of applications and computers mathematically to holistically optimize the whole system stack. Performance models of computer applications and libraries express their resource demands while performance models of computer systems describe their capabilities. Applications, middleware, and architectures can be co-designed by finding an optimal match of the machine capabilities to the application requirements in order to minimize the overall system cost. This work spans multiple areas of computer science and I will illustrate three key directions in the following: modeling, programming, and architecture.

1 Introduction

Society relies on fast computers to satisfy the growing demands of important applications such as drug design, weather prediction, and big data analytics. Around 2005, frequency scaling seized due to physical constraints and subsequently single-processor performance growth stopped as well. Moore's law, which predicts an approximate doubling of the transistor count per chip every 18 months, continues to hold and industry introduces ever-growing multi- and manycore processors to achieve the needed performance scaling. This shifted the problem of exploiting performance from the computer architect to the programmer who now has to write parallel programs that are substantially more complex than

their serial counterparts. This trend is deemed to continue so that average programmers are required to program with higher parallelism and potentially heterogeneous systems.

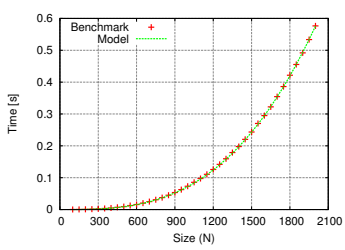
Today's state of the art in parallel programming is a collection of ad-hoc solutions: several libraries and languages offer various options for parallelization but developing efficient parallel software remains a complex process mastered by few programmers. We conjecture the main reason to be the complex interaction between parallel algorithm development, data location and communication, programming environments, and hardware details. We aim to untangle these relations with a new principled approach for parallel software design founded on mathematical modeling and optimization.

Our main research focus is to understand the interaction between architectures, programming models, applications, and cost (capital expenses, operational expenses, and time). In particular, we plan to develop tools based on solid theory to support the work of performance engineers and guide the design of future middleware and hardware systems. Our central technique is mathematical modeling and optimization of real-world systems in which we model architectural reality and trends either as constraints or cost-functions in the optimization space. This allows us to co-design algorithms, applications, and architectures by following optimizations in the abstract solution space. We expose simple performance models to the programmer to understand the behavior of application and system and we utilize more complex models internally for automatic optimizations.

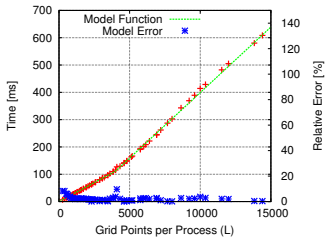
Three main research directions of our work have emerged as necessary for a successful outcome: (1) performance modeling where we strive to develop a comprehensive formalism for performance, (2) parallel programming models where we push for a complete change of perspective from control-centric to data-centric programming and models of computation, and (3) large-scale networking in which we apply model-based optimization techniques to real-world designs. We will now briefly outline our work in each of these directions.

2 Performance Modeling

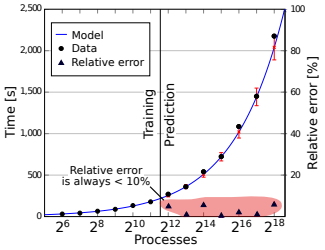
Performance modeling enables us to understand the application's requirements [8] and the architecture's features [14] in a rigorous analytic way. These models also enable model-driven co-design of applications and architectures and bottleneck analysis. Our main differentiating assumption is that hardware and software systems are too complex to be modeled ab initio. Thus, we use sets of predefined analytical hypothesis functions



(a) Execution time of tuned matrix multiplication (DGEMM) [8].



(b) Execution time of the GF kernel in the MILC application [1].

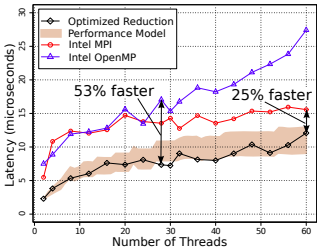


(c) Communication time of Sweep3D with $<10\%$ error [3].

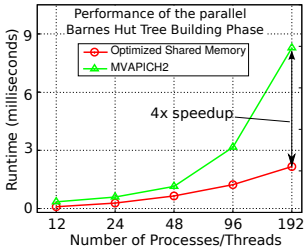
Figure 1: Examples for Performance Modeling. In (a) and (b), the crosses represent measurements and the lines represent analytical performance models. Figures are from the referenced full publications and credit goes to all respective co-authors.

combined with statistical techniques to select the right function and parametrize it to the exact system. Our semi-analytic performance modeling technique [8] generates models that can easily be interpreted by humans (e.g., for performance engineering) and which can be derived analytically to find minima or integrated to determine costs. We are in the process of automating these methods and apply them to new settings such as automatic compiler transformations.

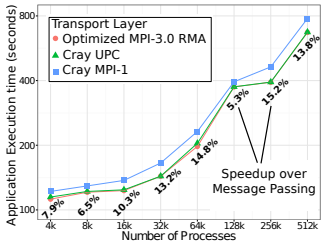
Performance modeling is most important to understand the performance characteristics of applications and architectures. In our view, a performance model is a function $f(\mathbf{x})$ that maps an application's input parameters (e.g., problem size) and environment (e.g., number of cores) as captured in the vector $\mathbf{x}$ to a resource requirement. One can generate performance models for different resources such as execution time, energy, or numbers of various operations. For example, the number of floating point operations for a straight-forward matrix multiplication for two matrices of size n is $f_1(n) = 2n^3$ while the execution time can be modeled with $f_2(n) = tn^3$ where t is a system-specific parameter that varies on each architecture and may be different for various ranges of n (in-cache and out-of-cache). We call these models semi-analytic models because they follow predefined analytic models (e.g., n^3) but are instantiated for each architecture (e.g., $t = 2.2ns$ on POWER7, see Figure 1a). Our method can be used to model complex applications including many kernels (cf. Figure 1b). We also demonstrated that the resulting models can be used for application tuning decisions [1]. At Supercomputing 2013 [3], we introduced a fully automated methodology for generating semi-analytic models for arbitrary HPC applications. Figure 1c shows an example result for the Sweep3D application. Such models can be used during all stages of the software engineering process: Analy-



(a) Model-driven optimization of broadcast on Xeon Phi [14].



(b) Performance gains by tuning shared memory [13].



(c) MILC application speedup using RMA programming [5].

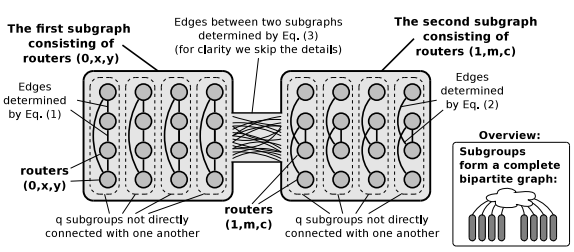
Figure 2: Examples of Programming Model optimizations. Figures are from the referenced full publications and credit goes to all respective co-authors.

sis (pick the correct method), Design (identify functional units and composition costs), Implementation (determine implementation tradeoffs such as cache blocking), Testing (check performance of implementation), Maintenance (lifetime performance monitoring to observe variations). Much of our recent work focuses on automating our developed performance monitoring principles by including static information from program analysis in the modeling process.

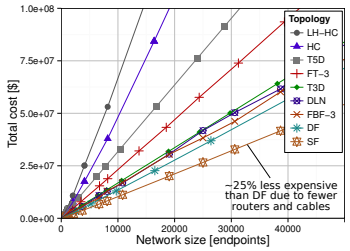
3 Parallel Programming

Parallel programming expresses parallelism and locality of applications. Most often, exposing parallelism is straight-forward while locality depends on the often intricate data-dependencies in applications. In our philosophy, we propose to change the view on programming from a control-centric to a data-centric model. Our main research focus in the past lied on parallel programming models, especially distributed memory parallel programming. The de-facto standard for distributed memory parallel programming is the Message Passing Interface Standard (MPI). MPI is supported on all supercomputers and is used for the vast majority of distributed-memory parallel programs. While MPI does not specify a programming model, its interface supports various different types of programming.

So far, we focused on the MPI-level where we enabled this view at the level of MPI processes [9, 11] and show its hardness. We also optimized data movement in the runtime system itself by developing message scheduling algorithms to exploit different system ar-



(a) Example Slim Fly MMS construction for $q = 5$.



(b) Total network cost.

Figure 3: The Slim Fly network topology; see Besta and Hoefler [2] for details.

chitectures [10, 13]. Model-driven design of such algorithms is the center of our research. Figure 2a shows the results of a model-based optimization of the broadcast operation on Intel's Xeon Phi architecture [14]. We developed a new hierarchical programming mode for MPI that allows to combine shared memory and MPI programming [6]. This model has already found several adopters and is taught in tutorials. Figure 2b shows potential performance gains by utilizing shared memory efficiently.

Traditional MPI programming uses the message passing concept. In MPI-3.0, we adapted the existing remote memory access (RMA) programming specification to modern remote direct memory access (RDMA) devices. Programming in this model can be more intuitive for some application and can result in significant performance gains [7]. Figure 2c shows application speedup for the MIMD lattice computation application using MPI-3.0 RMA on the Blue Waters Supercomputer. Yet, programming and modeling can only be as good as the underlying architecture. The last area we focus on is the large-scale networking used to connect tens of thousands of computers.

4 Large-scale Network Architecture

Large-scale networking is the main bottleneck for scalable high-performance parallelism. While the design of single cores is well understood, it is less clear how to connect multiple cores efficiently. Here, we work on network topologies as well as routing protocols utilizing mathematical cost and performance models. Many problems, such as deterministic network routing are NP-hard and we developed and implemented several effective heuristics. For example, our DFSSSP routing for InfiniBand [4] remains the fastest routing algorithm for irregular InfiniBand topologies today.

The most critical part of scalable parallel computer architectures are large-scale network topologies. The main development here is driven by technological capabilities of industry ("what can be built"). After the success of Dragonfly [12], we decided to apply mathematical optimization to the topology-design problem. Including all technological constraints, we were able to phrase network topology-design as an optimization problem. It turns out that the problem is similar to the long-studied degree-diameter problem. The designed topologies are up to 20% more cost- and energy-efficient than Dragonfly networks and have 33% less latency at comparable (full) bandwidth [2]. Our Slim Fly topology shown in Figure 3 is thus today's most cost-efficient and fastest known full-bandwidth network topology, and it is close to the optimal topology in our model.

5 Summary and Outlook

Parallel and high-performance computing is a quickly developing field. We chose to address the topics of performance modeling (to understand application performance mathematically), parallel programming (to understand principles of development of parallel applications), and network design (to understand principles of large-scale network architecture). We achieved to automate performance modeling, so far a manual process limited to a small group of experts. We introduced several new and promising programming concepts in MPI that can be adopted in a wider scope. And finally, we developed a new network topology that is cheaper than existing topologies and close to the optimum. We look forward with excitement to see new developments in the field of parallel and high-performance computing!

Acknowledgments

I am deeply grateful to the many mentors that helped me to guide my early research, especially Andrew Lumsdaine and Marc Snir. Since coming to ETH, I had the pleasure to interact with and be mentored by various faculty members who I thank. Markus Püschel deserves special credit for his ability to cheer me up in grim times and for improving me in many ways. I thank the Latsis foundation for their support of young scientists at all levels. Last but not least, I thank my students and colleagues for the great fun and collaborations we had and will have in the future. Thank you all!

References

[1] G. Bauer, S. Gottlieb, and T. Hoefler. Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application su3_rmd. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 652–659. IEEE Computer Society, May 2012.

[2] M. Besta and T. Hoefler. Slim Fly: A Cost Effective Low-Diameter Network Topology. Nov. 2014. Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC14).

[3] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf. Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. In *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC13)*, pages 45:1–45:12. ACM, Nov. 2013.

[4] J. Domke, T. Hoefler, and W. Nagel. Deadlock-Free Oblivious Routing for Arbitrary Topologies. In *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 613–624. IEEE Computer Society, May 2011.

[5] R. Gerstenberger, M. Besta, and T. Hoefler. Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided. In *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC13)*, pages 53:1–53:12. ACM, Nov. 2013.

[6] T. Hoefler, J. Dinan, D. Buntinas, P. Balaji, B. Barrett, R. Brightwell, W. Gropp, V. Kale, and R. Thakur. MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory. *Journal of Computing*, May 2013. doi: 10.1007/s00607-013-0324-2.

[7] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, and K. Underwood. Remote Memory Access Programming in MPI-3. *ACM Transactions on Parallel Computing (TOPC)*, Jan. 2015. accepted for publication on Dec. 4th.

[8] T. Hoefler, W. Gropp, M. Snir, and W. Kramer. Performance Modeling for Systematic Performance Tuning. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), SotP Session*, Nov. 2011.

[9] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, and J. L. Traeff. The Scalable Process Topology Interface of MPI 2.2. *Concurrency and Computation: Practice and Experience*, 23(4):293–310, Aug. 2010.

[10] T. Hoefler and T. Schneider. Optimization Principles for Collective Neighborhood Communications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 98:1–98:10. IEEE Computer Society Press, Nov. 2012.

[11] T. Hoefler and M. Snir. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*, pages 75–85. ACM, Jun. 2011.

[12] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Comput. Archit. News*, 36(3):77–88, June 2008.

[13] S. Li, T. Hoefler, and M. Snir. NUMA-Aware Shared Memory Collective Communication for MPI. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 85–96. ACM, Jun. 2013.

[14] S. Ramos and T. Hoefler. Modeling Communication in Cache-Coherent SMP Systems - A Case-Study with Xeon Phi. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 97–108. ACM, Jun. 2013.