Report from Dagstuhl Seminar 25341

Software Performance Engineering

Chen Ding[*1], Charles E. Leiserson[*2], Yihan Sun[*3], and Bruce Hoppe[†4]

1 **University of Rochester, US.** cding@cs.rochester.edu
2 **MIT – Cambridge, US.** cel@mit.edu
3 **University of California – Riverside, US.** syhlalala@gmail.com
4 **Connective Associates – Arlington, US.** behoppe@mit.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 25341 on software performance engineering. This seminar convened researchers from diverse intellectual communities across computer science to synthesize a collective understanding of this fragmented discipline and advance software performance engineering as a rigorous and principled scientific field in its own right. With connections established by this seminar, we are creating a unique arena for computer science researchers to cross-fertilize by sharing performance-engineering tools, techniques, opportunities, challenges, and open problems in their own domains of expertise. The major activities included 29 talks on recent research, five working groups that discussed topics like tools, community-building, education, and LLMs, and three "world cafe" sessions with in-depth conversations among participants on guiding questions for advancing software performance engineering.

Seminar August 17–22, 2025 – https://www.dagstuhl.de/25341
2012 ACM Subject Classification Computer systems organization; Computing methodologies; Software and its engineering; Theory of computation
Keywords and phrases applications, productivity tools, software performance engineering, theory and practice
Digital Object Identifier 10.4230/DagRep.15.8.1

1 Executive Summary

Chen Ding (University of Rochester, US)
Bruce Hoppe (Connective Associates – Arlington, US)
Charles E. Leiserson (MIT – Cambridge, US)
Yihan Sun (University of California – Riverside, US)

This seminar convened researchers from diverse intellectual communities across computer science who share a common interest in *software performance engineering* (SPE): making software run fast or otherwise consume few resources such as time, storage, energy, network bandwidth, etc. Seminar participants explored the role of SPE in each others' home fields of computer science and sought to synthesize a collective understanding of this fragmented discipline. The seminar aimed to coalesce a community of researchers to advance SPE as a rigorous and principled scientific field in its own right.

[*] Editor / Organizer
[†] Editorial Assistant / Collector

With the demise of Moore's Law, the time is ripe for this seminar. But despite the growing importance of SPE, research in the field is separated into tribes scattered across the traditional areas of computer science. Our major concern is how the field of computer science will fare without an SPE community that provides a common infrastructure, a shared understanding of SPE principles, and a lingua franca. We have a choice: SPE can be tedious, expensive, haphazard, and controlled by "high priests"; or it can be fun, cheap, principled, and democratically available to the average programmer.

With connections established by this Dagstuhl seminar, we are creating a unique arena for computer science researchers to cross-fertilize by sharing SPE tools, techniques, opportunities, challenges, and open problems in their own domains of expertise, with the aim of discovering commonalities that transcend individual tribes. We are promoting our arena for SPE with websites like `https://fastcode.org`, regular activities like the monthly Fastcode Seminar, and special projects like creating a SIMD tutorial with Highway.

2 Table of Contents

3 Overview of Talks

3.1 Transforming High-Performance Libraries to Domain-Specific Languages and Optimizing Compilers with BuildIt

Saman Amarashinghe (MIT – Cambridge, US)

URL https://buildit.so/

There are countless high-performance library implementations available for various domains and hardware platforms, yet Domain-Specific Languages (DSLs) and compilers remain rare. A well-designed DSL can express a far broader range of programs within a domain compared to even the most comprehensive library while also enabling domain-specific, global optimizations that go beyond hand-optimized kernels. The scarcity of high-performance DSLs stems from the complexity of building DSL compilers, which are typically large, intricate systems developed by experts.

In this talk, I will introduce BuildIt, a C++ framework designed for the rapid prototyping of high-performance DSLs. BuildIt uses a multi-stage programming approach to combine the flexibility of libraries with the performance and specialization of code generation. With BuildIt, domain experts can transform existing libraries into efficient, specialized compilers simply by modifying types of the variables. Moreover, it allows them to implement analyses and transformations without needing to write traditional compiler code. Currently, BuildIt supports code generation for multi-core CPUs and GPUs, with FPGA support coming soon. I will also showcase four DSLs created with BuildIt to highlight its power and ease of use: a reimplementation of the GraphIt graph computing language, the BREeze DSL for regular expressions, StreamIt a DSL for stream computing including PyTorch, and NetBlocks, a DSL for custom network protocol development. More information on BuildIt can be found at `https://buildit.so/`.

3.2 High-Performance Graph Analytics for Motif Finding in Neuroscience Connectome Graphs and Beyond using Arachne

David A. Bader (NJIT – Newark, US)

The growth of network-structured data across domains like neuroscience and cybersecurity demands scalable graph analytics, but complex tasks like subgraph isomorphism remain accessible only to high-performance computing (HPC) specialists. Arachne is an open-source framework that democratizes high-performance graph analytics through a Python interface while abstracting parallelism complexities. It enables advanced graph algorithms to run efficiently from laptops to supercomputers. Arachne has been adopted by Harvard researchers for the MoMo connectome visualization tool, allowing neuroscientists to draw neural motifs that are translated into attributed subgraphs and searched using our novel HiPerMotif algorithm. Key innovations include HiPerMotif, which achieves up to 66× speedups over parallel approaches.Testing on large-scale datasets including FlyWire and the H01 human brain connectome demonstrates Arachne's performance: completing complex subgraph

searches in 38 seconds versus NetworkX's 16,000+ seconds. This unified platform balances high-performance computation with accessibility, enabling researchers to extract insights from billion-scale graphs and advancing pattern matching across data-driven sciences.This research is supported in part by NSF grants CCF-2109988, OAC-2402560, and CCF-2453324.

3.3 Which RDF database is the best

Hannah Bast (Universität Freiburg, DE)

RDF is a modern variant of relational databases, with a universal schema (all data is modelled as triples) and universal identifiers (think of URLs). The query language is SPARQL, which is essentially SQL on RDF data. There are hundreds of fully-featured systems for RDF/SPARQL, from companies, open source projects, and academia. RDF and SPARQL are 100% standardized, yet the performance of these systems varies by many orders of magnitude, often contrary to the claims by the respective creators. In my talk, I will shed some light on this mystery. In a nutshell: computer scientists cannot code, open source developers slept in their algorithms class, and companies eventually stop doing research.

3.4 Lock-Free Locks

Naama Ben-David (Technion – Haifa, IL)

Locks are frequently used in concurrent systems to simplify code and ensure safe access to contended parts of memory. However, they are also known to cause bottlenecks in concurrent code, leading practitioners and theoreticians to sometimes opt for more intricate lock-free implementations. In this talk, I'll show that, despite the seeming contradiction, it is possible to design practically and theoretically efficient lock-free locks; I'll present a practical lock-free lock algorithm and our library which allows easily replacing regular locks with lock-free ones. Time permitting, I'll discuss related open problems.

3.5 Performance Engineering in a Legacy System: A Report from the Trenches

Jon Louis Bentley (Hackettstown, US)

Joint work of Jon Louis Bentley, Duffy Boyle, P. Krishnan, John Meiners

This talk describes how I spent six months embedded in an industrial-strength software team working as a consulting performance engineer. The system was big (>107 lines of code written by several hundred developers), aged (grown over several decades), important ($300M annual revenue) and slow – if our team couldn't speed it up, this cash cow and its revenue

would be catastrophically lost in a matter of months. I'll describe some of the problems that we faced, and challenge teachers of performance engineering to prepare their students to work in such a role. I'll also address how performance engineers can enhance their own performance by using LLMs. (This talk describes joint work with Duffy Boyle, P. Krishnan and John Meiners.)

3.6 SPE Expedition: Teaching Performance Engineering Through Collaborative Problem Solving

Rezaul Chowdhury (Stony Brook University, US)

This talk proposes a collaborative approach to teaching performance engineering. Building on the success of the long-standing Algorithms Reading Group at Stony Brook University, the proposed Performance Engineering (PE) meetings aim to foster a hands-on, interactive learning environment. Participants will collaboratively optimize unoptimized code through real-time coding, successive refinements, and discussions of interesting developments. The bi-weekly hybrid sessions will be open to all, recorded, and scribed, with opportunities for participants to continue optimizations between meetings and submit their code to a dedicated server for performance evaluation. An online leaderboard will track successful submissions, encouraging friendly competition and continuous improvement.

3.7 Stencil Computation with Reduced Work

Rezaul Chowdhury (Stony Brook University, US)

Joint work of Rezaul Chowdhury, Zafar Ahmad, Russell Bentley, Reilly Browne, Rathish Das, Pramod Ganapathi, Aaron Gregory, Yushen Huang, Michael Santomauro, Yimin Zhu

Stencil computations are a fundamental tool in scientific computing, modeling the evolution of physical systems on multidimensional grids over multiple timesteps. They underpin numerous applications including fluid dynamics, image processing, mechanical engineering, cellular automata, electromagnetics, and meteorology. Traditionally, stencil algorithms iterate over every grid cell for each timestep, performing Omega(TN) work for a grid of size N and T timesteps.

In this talk, I will present our recent sub-Theta(TN)-work algorithms for general linear stencil computations under aperiodic and free-space boundary conditions, achieving substantial speedups over state-of-the-art methods. Our aperiodic-grid algorithm employs an FFT-based periodic solver within a recursive divide-and-conquer framework (SPAA'21, TOPC), while our free-space algorithm uses Gaussian approximations and N-body methods (SPAA'22, ACDA'25).

More recently, we have extended our FFT-based approach to handle a broad class of problems involving multiple time-varying linear stencils applied across spatial regions with morphing boundaries, without increasing the work or span by more than a logarithmic factor in T (SPAA'25). We have also developed FFT-accelerated algorithms for a class of nonlinear stencil problems in quantitative finance, specifically for pricing American options under binomial, trinomial, and Black–Scholes–Merton models (PPoPP'24).

Implementations of most of these algorithms run significantly faster than the best existing methods.

This is joint work with current and former Stony Brook University students: Zafar Ahmad, Russell Bentley, Reilly Browne, Rathish Das, Pramod Ganapathi, Aaron Gregory, Yushen Huang, Michael Santomauro, and Yimin Zhu.

3.8 Successes and challenges in RocksDB performance at Meta

Peter C. Dillinger (Meta – Menlo Park, US)

RocksDB is a library providing a convenient key-value interface for data storage and sits on top of a local or remote filesystem. It is "ource of truth" for most of the online small-to-medium object storage at Meta. We have been involved in developing some advanced data structures such as relaxed consistency concurrent hash tables (unpublished HyperClockCache) and approximate query filters / static functions (Ribbon filter). We are looking to advance these and related areas, including other mutable concurrent data structures and other static representations of indexing or filtering data.

3.9 Relational and Complexity Theories of Locality

Chen Ding (University of Rochester, US)

Computer memory is hierarchical, making locality a fundamental concern in software performance engineering. Locality theories formalize the cost of a cache hierarchy. The formalism is necessary for optimization.

This talk overviews two recent theories of locality: the Relational Theory (Yuan et al., TACO 2019), which proves the equivalence of data movement, data reuse, and working set measures; and Data Movement Distance (Smith et al., ICS 2022), a new metric that quantifies locality by its asymptotic complexity.

3.10 Rank-polymorphism and performance: have your cake and eat it, too, with persistent asynchronous adaptive specialization

Clemens Grelck (Friedrich-Schiller-Universität Jena, DE)

Rank-polymorphic array programming systematically abstracts from structural array properties such as shape and rank. As usual, abstraction is great for software engineering, but comes at the price of performance. Specialization to rank- and shape-monomorphic intermediate code is the way out of this dilemma, but is limited in practice by the non-availability of rank and shape information at compile time.

We present a staged approach where the runtime system recompiles polymorphic intermediate code concurrently with the execution of the application itself. The resulting fast(er) code is then dynamically linked into the running application, and the runtime system, henceforth, dispatches program execution to the fast clone. Asymptotically, this yields shape-monomorphic performance for rank-polymorphic code.

We further complement asynchronous adaptive specialization with a persistence layer that stores fast clones in a repository for immediate use in further application runs that happen to use the same ranks and shapes. However, what sounds like a simple engineering solution turns out to exhibit a number of critical issues that we will briefly touch upon.

3.11 Recent Advances and Challenges in Parallel Algorithm Design

Yan Gu (University of California – Riverside, US)

With the advent of modern hardware, parallelism has been more important than ever, and top-tier conference papers reporting performance results are rarely run sequentially. Parallel algorithms have been extensively studied since the 1970s, so what's new and still needs to be explored? In this talk, I argue that there are still numerous important directions to investigate. I will briefly overview some of my recent work on graph analytics (SSSP, connectivity, k-core, etc.), data structure design (search trees, priority queues, kd-trees, etc.), and highlight ongoing challenges such as space-efficiency, synchronization costs, and the need for simplicity. If time permits, I will also discuss some future topics that may be of interest to this audience.

3.12 Compiler Technology for Multi-Scale Heterogeneity: a Data-Centric View

Mary W. Hall (University of Utah – Salt Lake City, US)

We need compilers that support the increasingly heterogeneous landscape of architectures. The end of Moore's Law and Dennard scaling has given rise to architecture specialization at multiple scales, from heterogeneous chip architectures that include chiplets and accelerators to integrated CPU+GPU nodes and heterogeneous clusters of heterogeneous nodes. This talk will focus on the role of data layout in generating code for all of these scales of heterogeneity.

3.13 Locating software performance engineering for the greater good

Bruce Hoppe (Connective Associates – Arlington, US)

The proposal for the Dagstuhl SPE seminar states: "We have a choice: SPE can be tedious, expensive, haphazard, and controlled by "high priests"; or it can be fun, cheap, principled, and democratically available to the average programmer." I call this "choosing where and how to locate SPE." In this talk, I introduce Fastcode – an open-source community for choosing where and how to locate SPE for the greater good.

3.14 LiBox: A Learned Index with Array-Like Efficiency and No Last-Mile Search

Song Jiang (University of Texas at Arlington, US)

Learned indexes can outperform traditional structures in speed and space efficiency by predicting key positions in sorted arrays. However, prediction errors necessitate costly last-mile searches, and model evaluation itself can be expensive, limiting performance gains. We present LiBox, a hierarchical, box-based learned index that eliminates these inefficiencies. LiBox partitions keys into "boxes" such that the target box can be identified with zero error using a simple linear regression function. The remaining in-box search requires only a single AVX-512 instruction, enabling highly predictable and minimal instruction counts per query. By using modest extra space to accommodate irregular key distributions, LiBox supports both read and write queries at near array-access speed. Its structure adapts reorganizations to workload patterns, sustaining high read performance while hiding update costs.

3.15 Speedcode: Software performance engineering education via the coding of didactic exercises

Timothy Kaler (MIT – Cambridge, US)

URL https://speedcode.org

This talk discusses recent work to develop structured playgrounds for learning about software performance engineering. Speedcode is an online programming platform that aims to improve the accessibility of software performance-engineering education. At its core, Speedcode provides a platform that lets users gain hands-on experience in software performance engineering and parallel programming by completing short programming exercises. Speedcode challenges users to develop fast multicore solutions for short programming problems and evaluates their code's performance and scalability in a quiesced cloud environment.

I will also discuss the development of FastCoder-Factory and Lesson which relate to synthetic data generation and collaborative learning for code-optimization tasks.

3.16 Talk: Portable Compilation in an Accelerated World

Fredrik Kjolstad (Stanford University, US)

The future of software performance engineering must include the specialized hardware that is being developed across the industry and academia. Although performance engineers play a critical role, such hardware places a large burden on the software stack and thus increases the need for compilers and programming models for productivity. I will share my thoughts on designing programming systems that permit portable compilation across disparate hardware. These programming systems must raise the level of abstraction to diverse operations on abstract collections. (I think four such collections cover the lion's share of large-scale computing.) By raising the level of abstraction and by introducing new compiler techniques, we can make programs portable across different machines and different data structures. To manage complexity, compilers should target hardware-facing abstract machines that separate the software and hardware implementations. Finally, intermediate languages can also help us describe hardware to the compiler, so that we can target it without rewriting large parts of the compiler.

3.17 Dynamic Partial Deadlock Detection and Recovery via Garbage Collection

I-Ting Angelina Lee (Washington University – St. Louis, US)

A challenge of writing concurrent message-passing pro- grams is ensuring the absence of partial deadlocks, which can cause severe memory leaks in long-running systems. The Go programming language is particularly susceptible to this problem due to its support of message passing and ease of lightweight concurrency creation.

We propose a novel dynamic technique to detect partial deadlocks by soundly approximating liveness using the garbage collector's marking phase. The approach allows systems to not only detect, but also automatically redress partial deadlocks and alleviate their impact on memory.

We implement the approach in the tool Golf, as an extension to the garbage collector of the Go runtime system and evaluate its effectiveness in a series of experiments. Preliminary results show that the approach is effective at detecting 94% and 50% of partial deadlocks in a series of microbenchmarks and the test suites of a large-scale industrial codebase, respectively. Furthermore, we deployed Golf on a real service used by Uber, and over a period of 24 hours, effectively detected 252 partial deadlocks caused by three programming errors.

3.18 Towards Zero Spawn Overhead: Work Stealing Without Deques

I-Ting Angelina Lee (Washington University – St. Louis, US)

In a randomized work-stealing scheduler, parallel speedup depends on the spawn overhead, which workers pay to allow tasks to execute in parallel, and the steal overhead, which thieves pay to start executing new work. It's important to minimize the spawn overhead, because the spawn overhead incurred by the parallel code must first be offset by parallel scalability before any speedup can be observed.

In pursuit of zero spawn overhead, this work considers a strategy that eliminates the use of deques entirely, obviating the need for a worker to perform explicit bookkeeping or set up a deque to enable parallelism. To that end, we propose DLite, a compiler and runtime ABI (Application Binary Interface) that incurs near-zero spawn overhead, empirically measured to be about 6% compared to a regular function invocation. DLite decreases the spawn overhead to almost nil, at the expense of a high steal cost. Specifically, DLite employs a backtracking strategy: When a steal attempt occurs, the victim provides its current stack and base pointers to the thief, and the thief then reconstructs the necessary state to realize the parallel execution.

We have implemented Cilk-DLite, which extends the OpenCilk platform to implement DLite. When the application has ample parallelism, Cilk-DLite exhibits similar scalability to OpenCilk with much lower spawn overhead. When the application lacks parallelism, the high steal cost in Cilk-DLite can impede scalability due to slower work distribution. We deep dive into one benchmark to analyze the performance impact of the high steal cost.

3.19 The resurgence of software performance engineering

Charles E. Leiserson (MIT – Cambridge, US)

Today, most application developers write code without much regard for how quickly it will run. Moreover, once the code is written, it is rare for it to be reengineered to run faster. Historically, gains in performance from miniaturization, codified in Moore's Law, relieved programmers from the burden of making software run fast. But with the end of Moore's Law, interest is resurging in software performance engineering: making software run fast or otherwise consume few resources such as time, storage, file IO's, network bandwidth, energy, etc. In the future, to develop innovative products and applications, programmers will need to engage in performance engineering, which is an integrative field requiring an understanding of algorithms, software, and computer architecture. Unfortunately, performance engineering is not widely taught, and most people consider it an unstructured collection of ad hoc tricks. Now is the time to establish software performance engineering as a science-based discipline in its own right alongside traditional areas of computer science.

3.20 Dataflow-Specific Algorithms for Resource-Constrained Scheduling and Memory Design

Quanquan C. Liu (Yale University – New Haven, US)

We introduce the Weighted Red-Blue Pebble Game, an extension of the classic red-blue pebble game with weighted operation costs. This weighted formulation enables constant-factor analysis of highly resource-constrained systems with bounded fast memory, unlimited slow memory, and strict energy and power constraints.

We apply our model to computational kernels in ultra-low-power brain-computer interfaces (BCIs) implanted near the brain. We express these kernels as computational directed acyclic graphs (CDAGs), enabling modular composition of operation schedules with data movement. We derive theoretically optimal schedules for a broad class of tree-structured CDAGs and apply them to on-chip memory design with circuit-level validations for power and area.

Our algorithms result in an average 63% memory area reduction and 43% static power reduction for BCI workloads–critical improvements for ensuring safe, thermally constrained operation in implantable devices. Beyond BCIs, our results underscore the broader utility of weighted pebble games in optimizing memory and I/O across resource-constrained computing environments.

3.21 HPCToolkit: A Tool for Application Performance Engineering at Scale

John Mellor-Crummey (Rice University – Houston, US)

URL https://hpctoolkit.org/

HPCToolkit is a tool for performance analysis of programs on systems ranging from desktops to GPU-accelerated supercomputers. Hardware support for instruction-level performance measurement in AMD, Intel, and NVIDIA GPUs was developed at the urging of the HPCToolkit project team. When measuring a GPU-accelerated application, HPCToolkit employs novel wait-free queues to communicate performance measurements between tool threads and application threads. To accelerate attribution of an execution's performance measurements, HPCToolkit analyzes the execution's CPU and GPU binaries to recover mappings between machine instructions and source code. To analyze terabytes of performance measurements gathered during executions at exascale, HPCToolkit employs distributed-memory parallelism, multithreading, sparse data structures, and out-of-core streaming algorithms. To support interactive exploration of profiles up to terabytes in size, HPCToolkit's hpcviewer GUI uses out-of-core methods to visualize performance data. Recently, HPCToolkit was extended with support for top-down CPU performance analysis. This talk will describe key aspects of HPCToolkit, successes analyzing applications, and some challenges ahead.

3.22 Making Waves in the Cloud: A Paradigm Shift for Scientific Computing through Compiler Technology

William S. Moses (University of Illinois – Urbana-Champaign, US)

Scientific models are today limited by compute resources, forcing approximations driven by feasibility rather than theory. They consequently miss important physical processes and decision-relevant regional details. Advances in AI-driven supercomputing – specialized tensor accelerators, AI compiler stacks, and novel distributed systems – offer unprecedented computational power. Yet, scientific applications such as ocean models, often written in Fortran, C++, or Julia and built for traditional HPC, remain largely incompatible with these technologies. This gap hampers performance portability and isolates scientific computing from rapid cloud-based innovation for AI workloads. In this work, we bridge that gap by transpiling existing programs using the MLIR compiler infrastructure. This process enables advanced optimizations, deployment on AI hardware, and automatic differentiation. In particular, we demonstrate execution of a state of the art Julia-based ocean model (Oceananigans), with >277 custom single-node CUDA kernels on thousands of distributed GPUs and Google TPUs. Our results demonstrate that cloud-based hardware and software designed for AI workloads can significantly accelerate simulations, opening a path for scientific programs to benefit from cutting-edge computational advances.

3.23 Zombie Hashing Reanimating Tombstones in a Graveyard

Prashant Pandey (Northeastern University – Boston, US)

Linear probing-based hash tables offer high data locality and are considered among the fastest in real-world applications. However, they come with an inherent tradeoff between space efficiency and speed, i.e. when the hash table approaches full capacity, its performance tends to decline considerably due to an effect known as primary clustering. As a result they are only used at low load factors.

Tombstones (markers for deleted elements) can help mitigate the effect of primary clustering in linear probing hash tables. However, tombstones require periodic redistribution, which, in turn, requires a complete halt of regular operations. This makes linear probing not suitable in practical applications where periodic halts are unacceptable.

In this talk, we present a solution to forestall primary clustering in linear probing hash tables, ensuring high data locality and consistent performance even at high load factors. Our approach redistributes tombstones within small windows, deamortizing the cost of mitigating primary clustering and eliminating the need for periodic halts. We provide theoretical guarantees that our deamortization method is asymptotically optimal in efficiency and cost. We also design an efficient implementation within dominant linear-probing hash tables and show performance improvements.

We introduce Zombie hashing in two variants: ordered (compact) and unordered (vectorized) linear probing hash tables. Both variants achieve consistent, high throughput and lowest variance in operation latency compared to other state-of-the-art hash tables across numerous

churn cycles, while maintaining 95% space efficiency without downtime. Our results show that Zombie hashing overcomes the limitations of linear probing while preserving high data locality.

3.24 Concurrent Size

Erez Petrank (Technion – Haifa, IL)

The size of a data structure (i.e., the number of elements in it) is a widely used property of a data set. However, for concurrent programs, obtaining a correct size efficiently is non-trivial. In fact, the literature does not offer a mechanism to obtain a correct (linearizable) size of a concurrent data set without resorting to inefficient solutions, such as taking a full snapshot of the data structure to count the elements, or acquiring one global lock in all update and size operations. I will talk about a methodology for adding a concurrent linearizable size operation to sets and dictionaries with a relatively low performance overhead.

3.25 Rank-Polymorphism for High-Level Performance Engineering

Sven-Bodo Scholz (Radboud University Nijmegen, NL)

URL https://www.sac-home.org/

Rank-Polymorphism relates to the ability of specifying algorithms that operate on arrays of statically undetermined dimensionality and shape. As it turns out, this capability is instrumental for enabling various performance-related aspects such as parallelism and locality to be captured in array shapes rather than explicit loop nests. At the example of SaC (www.sac-home.org), we demonstrate how this can be achieved, delivering competitive performance from very concise, easily verifiable, rank-polymorphic specifications.

3.26 Automating Performance Optimization of Data Flow Within HPC Workflows

Nathan Tallent (Pacific Northwest National Lab. – Richland, US)

Scientific workflows that require HPC resources are critical in many areas of scientific exploration. Because these workflows tend to be data intensive, severe bottlenecks emerge in storage systems and I/O networks. Although there has been much prior work on coordination of workflows, scheduling algorithms, and HPC storage systems, there are no comprehensive workflow performance diagnosis suites that can automatically identify and alleviate bottlenecks. We present DataFlowDrs, a new comprehensive suite of tools for performance optimization of HPC workflows that especially focuses on data flow and storage. Our suite

introduces (a) lightweight high-resolution tools for measurement and profiling of executions; (b) novel methods for automatically predicting data flow bottlenecks after only a 3-5 runs using automatically generated interpretable models of data flow; (c) effective performance analysis and bottleneck detection that can automatically rank order bottlenecks for different combinations of task parallelism and storage resources; (d) actionable performance optimization in the form of new schedules and resource assignments.

3.27 Memory Systems Challenges in Graph Processing

Hans Vandierendonck (Queen's University of Belfast, GB)

Graph analytics are reputed for posing significant challenges to the memory system, which is primarily due to "random" memory access patterns. In this talk, we revisit performance models and characterisation of the memory system issues. We discuss two solutions to the problem: vectorisation and data compaction and evaluate how they alleviate the bottleneck presented by the memory system.

3.28 Parallelism-corrected profiling

Jan Wassenberg (Google – Zürich, CH)

Amdahl's Law – that serial sections ultimately limit parallel speedup – is often obscured by modern profilers. In fork-join parallelism, standard tools misrepresent execution costs, allowing serial bottlenecks to hide in plain sight. We introduce Parallelism-corrected profiling, a straightforward method to adjust profiler outputs according to their "wall time" contribution. We apply this method to LLM inference on CPU, finding several bottlenecks that resulted in an easy 1.4x speedup.

3.29 Benchmarking and developing dynamic-graph data structures

Helen Xu (Georgia Institute of Technology – Atlanta, US)

This talk will cover everything you need to know about how to design efficient parallel applications that operate on dynamic graphs! It will focus on three key aspects:(1) How do you design the containers (ie, data structures) that encapsulate the dynamic graph? (2) What is the right framework (ie, interface) for interacting with a dynamic graph? (3) How do you fairly benchmark the performance of your parallel application for dynamic graphs?

To answer these questions, this talk will discuss two main results. First, I will present a new container for dynamic graphs called F-Graph. F-Graph is a multicore batch-parallel dynamic-graph system that is optimized for spatial locality. It is built on top of a batch-parallel packed-memory array, yielding fast performance for a variety of graph applications.

Next, I will present BYO, a unified framework for large-scale graph containers designed to facilitate benchmarking. BYO provides a simple and abstract container API, along with a clean interface. The evaluation uses BYO to evaluate 27 different graph containers on 10 different graph algorithms using 10 large graph datasets. The resulting data illuminates the issues and tradeoffs involved in designing parallel applications for dynamic graphs. Overall, the talk hopes to provide insight into both the theory and practice of efficient parallel computation for dynamic graphs.

4 Working groups

4.1 SPE and LLMs

Saman Amarashinghe (MIT – Cambridge, US), David A. Bader (NJIT – Newark, US), Jon Louis Bentley (Hackettstown, US), Albert Cohen (Google – Paris, FR), Timothy Kaler (MIT – Cambridge, US), Charles E. Leiserson (MIT – Cambridge, US), and Quanquan C. Liu (Yale University – New Haven, US)

4.1.1 ML Impact on Performance Engineering

Discussion of topics for further discussion

- Popular topics emerged from voting:
  - How will software performance engineering (SPE) change/improve with LLMs?
  - What are the LLM-related research topics for SPE?
- Group split into two discussion groups focusing on these areas
- Initial demonstration showed live coding with LLM generating a 400+ line app in minutes

What are the LLM-related research topics for SPE

- Role of correctness in LLM-generated code:
  - Shift from generating correct code by design to verifying correctness after generation
  - Two approaches debated:
    - Use existing tools to constrain LLMs and increase correctness probability
    - Let LLMs "go wild" with superoptimization, then verify afterward
- Tool integration questions:
  - Whether existing performance engineering tools remain relevant in LLM environment
  - Scheduling languages (like Mary Hall's morning presentation) could become LLM vocabulary
    - LLMs could generate schedules that compose into correct, high-performance code
    - Domain-specific languages (DSLs) designed for machine consumption
- Transferability research opportunities:
  - LLMs translating optimized code between languages (e.g., C/OpenMP to Rust)
  - Challenge: very few parallel code bases exist for training
  - Transpiler research evolution from syntactic to semantic approaches
- Expanded scope beyond LLMs to general ML approaches

Common thread: correctness

- Central challenge: making it easier for ML to generate correct code
  - LLMs can generate code faster than verification tools can check it
  - Much faster generation than humans can comprehend or analyze
- Consensus that LLMs cannot guarantee correctness in foreseeable future
  - Verification will likely remain harder than code generation
  - Need for dynamic correctness checking beyond static analysis
- Bridge approach proposed: use current tools to guide LLMs without complete restriction

LLM can generate code

- Speed advantages:
  - Faster than correctness tools can verify
  - Much faster than humans can comprehend
- Agreement that trusting LLM correctness remains problematic
- Resources for staying current:
  - Martin Maas blog (though may not be recently updated)
  - ML for systems conferences and SysML conferences

What are LLMs good for

- Performance diagnosis and analysis:
  - Processing complex performance traces and providing optimization suggestions
  - Identifying serialization bottlenecks in large-scale traces
  - Acting as intelligent performance analysis assistant
- Autotuning replacement potential
- Research assistance:
  - "DeepResearch" mode for literature review
  - AI co-scientist frameworks for hypothesis generation
  - Example: Google DeepMind's multi-agent system identifying research directions
- Hypothesis generation and literature synthesis:
  - Processing 200+ papers quickly for initial analysis
  - Generating ranked research direction recommendations

What SPE can do for LLM?

- Jan Wassenberg identified as expert resource for this perspective
- Topic noted but not extensively discussed in this session

Tools to help us

- Research tools mentioned:
  - Research Rabbit: citation graph tracing and clustering
  - Notebook LM: high-confidence summarization with papers loaded in context
  - Consensus AI: survey and research assistance
- Limitations noted:
  - Research Rabbit includes tangentially related papers
  - Need for careful curation before feeding into analysis tools

Questions

- Most interesting/productive use of LLMs for SPE research priorities
- Research topics to explore in ML for systems community
- Correctness approaches:
  - Design-time correctness vs. post-generation verification
  - Integrating theorem provers with LLM workflows
- Promising applications:
  - LLMs for performance data analysis
  - Compiler error message interpretation and debugging assistance
  - Survey paper generation as starting point for human refinement

4.1.2 Roundtable Discussion Notes

How would you use LLMs for coding; how do you get started?

1. Tim: use $10 credit on OpenAI playground and play around with prompts.
   a. Try Cursor. To study the models, get the API for OpenAI.
2. David: all models have strengths and weaknesses. Pay for the more advanced models, not just use the free version; the paid version will be much better.
   a. Mileage may vary with each model and version number.
   b. There may be a meta-model that calls all other models.
   c. I'm sold on Anthropic Claude, does better than ChatGPT.
   d. Deepseek is quite good as well.
3. Jon: What was the nature of the prompts? Context offers a great deal.
4. Saman: What's the difference between using interfaces (on the web) and agents?
   a. Tim: Can create custom workflows with agents. Now I'm using Cursor and I don't even need to make a custom workflow. For unstructured things like analyzing data or asking a topic I don't have any expertise in, I use OpenAI Playground. The interesting workflow right now is just to use Cursor. Iterate through generating and testing.
   b. David: Copy and paste from emacs.

What do you usually do using LLMs?

1. Saman: Can you write me an app for voting for topics to discuss for this discussion session? It did two things wrong. Then I put screenshots stating the problems. Then, it didn't start. Then, I said make these three changes. Then, it worked.
2. David: how do we benchmark the LLM-generated code? What is a systematic way for benchmarking the LLMs to each other? Is there a way to systematically check all combinations of optimizations? Is there a way to check all possible combinations of optimizations?
3. Jon: I've talked to LLMs to make summaries and transfer the summaries from one context to another. Can refer to details about everything.
4. Saman: We were trying to write a full Apple app. At some point, we ran out of context and then we couldn't get past it.
5. Jon: Gemini has much more context than ChatGPT; 1 million tokens versus 50k which makes a big difference.
6. David: We've used Rag to deal with context issues.
7. Jon: Use case: read a book and asked what are the lessons for software engineers from this book?

8. Tim: LLMs are particularly good at acting as experts. They are particularly good at being experts at AI.
   a. Use cases: using LLMs for productivity tools. You can use LLMs to evaluate LLMs using these tools. You can use them for user evaluations. It's an objective evaluation of productivity tools.
   b. It's good to have formal verification but people don't want to write these formal verification languages.
9. Saman: It'll be great to write a formal verification proof for transforms. Give a proof that the software is correct.
   a. David: We still need a verifier for verifying the formal verification proof for transforms.
10. Saman: can we use LLM fine-tuning to train on machine code and have it output machine code?
11. Jon: good blog posts: a) use cases for SPE b) what can you use LLMs in real life.
    a. Saman: 5 use cases for LLMs that people in our community have used it for.
12. Jon: give a technical paper to an LLM and let it give critique for the paper.
13. Saman: how do I improve my chances for acceptance? They gave a lot of good ideas.
14. Long discussion on using LLMs for recommendations and bios.
15. Tim: if there's something that's verification, you can use reinforcement learning to fine-tune the LLM.

Looking into the future

16. Jon: in the future will there still be books? Or will LLMs supercede books?
17. Saman: using LLMs, you don't have to go through the middle part about understanding why an answer is correct. All previous generations have to go through this training. Every generation, you lose some abilities.
18. Charles: but what do you get back?
19. David: I think in 2 or three years, all coding will be done in English.

4.2 SPE Education (Curriculum and Teaching)

Naama Ben-David (Technion – Haifa, IL), Saman Amarashinghe (MIT – Cambridge, US), Rezaul Chowdhury (Stony Brook University, US), Bruce Hoppe (Connective Associates – Arlington, US), Song Jiang (University of Texas at Arlington, US), Timothy Kaler (MIT – Cambridge, US), Fredrik Kjolstad (Stanford University, US), Charles E. Leiserson (MIT – Cambridge, US), Nikolai Maas (KIT – Karlsruher Institut für Technologie, DE), Manuel Penschuck (Goethe University – Frankfurt am Main, DE), Hans Vandierendonck (Queen's University of Belfast, GB), Marvin Williams (KIT – Karlsruher Institut für Technologie, DE), and Helen Xu (Georgia Institute of Technology – Atlanta, US)

4.2.1 SPE Curriculum

Context: MIT 6.106, course on SPE, is restricted to multi-core (parallelism). It is not: concurrency; distributed systems; databases; networking; etc. However, students are trained in general ideas, not specifics of each of the different areas, which they can acquire on the job.

Observation: multi-core course sufficient as initial course for undergraduates.

Summary of discussion

- Consensus on core set of SPE curriculum: experimental skills on performance measurement, tools enabling detective-work, examples of relevant optimisations
- Parallel programming may have to take precedence over single-core optimisation to make memory system optimisations relevant on many-core processors
- Advanced courses can branch out in different directions and are largely independent, e.g., accelerators, file systems, data bases, distributed systems

4.2.2 Teaching SPE

Who we are and why we are here

We all teach something closely related to SPE:
- Teaching Algorithm Engineering & related courses
- Wanting to offer new course on SPE
- Teaching DB systems & scalable algorithms course; starting course on SPE
- Already teaching course related to SPE

The fastcode SPE instructors community

`https://fastcode.org/get-involved/instructors/`

- offers helpful resources and a "meta-class" on teaching SPE
- Plan: discuss how to use the materials
- Valuable conversations about teaching are hard because of private political restrictions within each institution.

Opinions on MPI course

- in the past, no test harness or benchmark framework was provided
  - Try to use educational cluster, but has high variability -> e.g. due to other workloads
  - AWS is expensive and a lot of work
  - Fastcode has some infrastructure resources
    - speedcode coding platform
    - project on providing a script system is on the way
  - Is it possible to use a simulator or instruction counts? -> problematic if parallelism is involved
- It is valuable to offer resources which are immediately usable
- Students sometimes prefer to run code online instead of locally
  - It is possible to teach them how to debug locally by providing an introduction / instructions

Should we teach profilers / how to best do it

- start with a debugging assignment
- it is not really possible to force students to use a profiler, but you can encourage them
- competition on optimizing code (online leaderboard) helps

Curriculum: how does the basic course look like

- there is a page on fastcode.org with public slides
- there is a lot of different things that should be included in such a course
- using simulators (e.g. simulate cache misses in Python) can help to make concepts more accessible, especially for more theory-leaning students
- it can be valuable to show differences between theoretical expectations and practice

Experience from talking with practitioners

- systems they work on are severely restricted
- joy of SPE comes from finding things that work within such an environment
- how to share this with students?

It is important to teach about concepts / guiding principles

- instead of only showing how things work in practice (course on low-level languages vs. OS course)
- differentiate programming as a tool from the abstract concepts
- decoupling is really important in all fields in CS
  - but often loses performance -> SPE tends to break down some abstractions

4.3 Ten Questions of SPE

Jon Louis Bentley (Hackettstown, US)

Q1: What, precisely, is Software Performance Engineering (SPE)?

Please give a definition that is useful for deciding what work falls into this area. Various tests for any proposed definition.

- Is it broad enough to include all of the people at this workshop? If not, should it?
- Does it include the items in Q2 and Q3 below?

Do we need both a regular definition and a "Big Tent" definition? What is the relationship of SPE to other fields, including

- Compiler Optimization – is this a subfield of SPE?
- Algorithm Engineering – Is this identical to SPE?
- Software Engineering – Is SPE a subfield of SE?

What CS topics clearly are *not* part of SPE?

- For any field X, the efficient implementation of X objects falls within SPE.
- Excluding that connection, what CS fields are not part of SPE?
- A theory-only field, such as Axiomatic Complexity Theory

What is research in SPE? Is there research in SPE, or does SPE build on the research done in other fields? Potential definitions include the following components:

- *SPE is Engineering*: A branch of science and technology concerned with the design, building and use of computing systems.

- *SPE is about Performance*: Performance engineering is a systematic approach to optimizing the efficiency of software and systems throughout their entire lifecycle.
- SPE is about Performance Engineering *applied to software systems*.

One LLM Generated Definition: Software Performance Engineering (SPE) is a systematic, data-driven discipline focused on the design, measurement, and optimization of computing systems to meet defined performance goals throughout their entire lifecycle.

Q2: What are some classic readings in SPE?

Books

- Abbott & Fisher, The Art of Scalability
- Bentley, Writing Efficient Programs
- Kounev, Systems Benchmarking
- McGeoch, A Guide to Experimental Algorithmics
- Sites, Understanding Software Dynamics

Articles

- Knuth, An Empirical Study of Fortran Programs
- Bentley & McIlroy, Engineering a Sort Function

Q3: Give an overview, a map, of the field of SPE today

Questions Q1, Q2 and Q3 are intimately related. The responses to Q2 and Q3 will depend on the definition in Q1. The definition of Q1 can be tested by whether it includes the responses to Q2 and Q3.

Q4: What are the Fundamental Principles of SPE?

- Measure first, and measure often.
- Reality is the only judge.
- Understand the entire stack.
- Performance is a continuous process, not a one-time event.

Q5: What are the Essential Techniques of SPE?

(This is distinct but perhaps related to the Essential Tools of SPE)
Benchmarking as a technique...

Q6: What are the Ten Big Problems of SPE today?

What is a Big Problem? Do we mean "grand challenges" or do we mean "looming risks"? The two can be closely related: any obvious looming risk implies the grand challenge of mitigating that risk. Correctness is one such looming risk that implies a grand challenge

Q7: How should an SPE determine when to stop optimizing?

What are different ways of setting performance goals? How to choose how much effort to expend in increasing performance?

- How to estimate what future performance requirements will be?
- How to choose safety factors?
- How to balance human productivity and machine performance?

List different criteria for stopping the performance improvement process.

Q8: What is the value of SPE? Please illustrate with examples.

This is easy to see when things go wrong, but how to illustrate the value when everything is going right? (The benefits of keeping healthy rather than curing illnesses.) What is there to lose if we do not invest in SPE?

Q9: What is the role of asymptotic analyses in SPE?

Probably not a goal in itself, but one of many useful tools.

QUESTIONS COVERED BY OTHER GROUPS

- What are the essential tools for an SPE?
- What roles can LLMs and other ML tools play in SPE? What should be the structure of an SPE undergraduate curriculum?
- Should there be a graduate SPE curriculum? If so, what should its structure be?

4.4 SPE Tools

John Mellor-Crummey (Rice University – Houston, US) and Jan Wassenberg (Google – Zürich, CH)

4.4.1 Diverse set of architectures

- Cerebras
  - Tiled architecture with 48KB memory per tile
  - CSL programming model supports HPC applications beyond deep learning
  - Student worked on targeting with custom applications, required vendor engagement to extend communication capabilities
  - G42 from Abu Dhabi made major investment to keep company viable
- Neuromorphic architectures
  - Most are actually just ML-targeted accelerators, not true neuromorphic
- Sambanova
  - Dataflow graph compilation model
  - Tooling would require compiler passes that decorate graphs for instrumentation
  - Programming model not publicly exposed
- GROQ
  - Intel expressed interest in targeting this architecture
  - Availability unclear for purchase
- Risk-V based accelerators
  - STX accelerator from Fraunhofer
    - Stencil Tensor accelerator with data management cores and compute engines
    - Two-level offloading model using bastardized OpenMP
  - LLVM extensions being developed

4.4.2 What kinds of tools

- Programming systems and compilers
  - Most vendors building proprietary MLIR compilers
  - Limited software accessibility beyond deep learning frameworks
- Runtime systems
  - Message passing and vector streaming communication models
  - Fine-grain partitioning required for tile architectures
- Performance tools
  - Simulators provide timing and memory contention information
  - Analysis tools need architecture-specific development
  - 80% of specialized hardware expected to disappear within couple years

4.4.3 Challenges

- Proprietary hardware dominance
  - Vendors unwilling to open source competitive advantages
  - Secret sauce optimizations kept internal
- F64 support disappearing
  - Requires analysis to determine when reduced precision acceptable
  - F64 emulation with F16 requires dozens of passes
  - Need tools to verify correctness of reduced precision results
- Most programming systems focused on machine learning
  - PyTorch, TensorFlow primarily ML-oriented
  - HPC applications struggle to find suitable tooling
- Industry vs open source divide
  - AI development unlike Internet era – no unified direction
  - Industry investment outpacing federal funding dramatically
  - Department of Energy focused on GPU-accelerated supercomputers, not novel architectures

4.4.4 Challenge: Open source tools

- Programming models seeking commonality
  - DSLs as vehicle for targeting multiple systems
  - Abstraction layers closer to application level needed
  - Rust interest for safe concurrent software development
- Promising approaches
  - **Triton**: Tile-level loop fusion programming model for GPUs
  - **PALLAS**
    - Embedded in JAX (Python)
    - References to array slices with explicit placement control
    - Can generate Triton code
    - Open source status confirmed
    - Higher level abstractions than Triton
  - **ONNX**: Common graph representation
    - Powerful enough to express non-ML computations
    - Still requires sophisticated partitioning for tile architectures
    - Need higher-level models to generate ONNX representations

- **JAX**:
  - JIT compilation for NumPy interface
  - Supports CPU, GPU, TPU platforms
  - Used at Google for weather simulation and other scientific computing
  - Allows custom code integration while bypassing autograd
- **Single Assignment C (SAC)**:
  - Generates C/CUDA code from high-level descriptions
  - Targets multiple architectures including potential new ones
- IR target options: LLVM + MLIR
  - LLVM becoming de facto standard (Intel, IBM, Google, Microsoft support Clang)
  - MLIR offers flexibility but fragmented into many dialects
  - Convergence toward MHLO dialect for high-level operations
  - MHLO used for non-ML applications (compute sin, cos operations)
  - Challenge: MLIR not designed for heterogeneous programming
  - Need explicit parallelism mapping from implicit dataflow graphs

(Note: StableHLO is apparently the preferred replacement for/derivative of MHLO.)

4.5 "Highlights of SPE" Workshop

Yihan Sun (University of California – Riverside, US), Yan Gu (University of California – Riverside, US), Rob Johnson (Broadcom – San Jose, US), and Prashant Pandey (Northeastern University – Boston, US)

Below are very brief notes of the discussions:

- **Name?** Highlights of Performance Engineering (HOPE) or Highlights of Performance Engineering on Software (HOPES)?
- **When/Where?** Possible plan: 2027 PPoPP/HPCA/CGO/CC/.. Co-located conferences on multiple relevant areas; good location (Salt Lake City); flexible timeline. It doesn't need to be always at PPoPP, though. Maybe we can reconsider the venue every year based on time/location/...
- **How to submit?**
  - Submissions will be based on "highlights", which are papers published elsewhere but highly relevant to SPE. Accepted papers will be invited to give talks.
  - We can make it submission+invitation based. The PC (or an even smaller committee) may recommend papers, but submissions are open to the public.
  - Papers will be evaluated by a 2-4 pages abstract. Full papers should be provided. Accepted papers will be invited to give talks; more papers may be accepted as posters. The quality of the write-up is important in the review process as it is an indicator of the talk quality.
  - A submission can contain multiple papers or a series of work. In this case the abstract should provide a nice overview of them.
  - Accepted manuscripts can be published as abstract/workshop short papers on ACM DL.

- **Other discussions**
  - Tracks? Expected number of submissions/acceptance rate? Balance topics?
  - Possible funding for student traveling? (would make it more attractive to get more submissions)
  - For invited papers, should we always accept or they still compete with other papers normally?

Participants

- Saman Amarashinghe
MIT – Cambridge, US
- David A. Bader
NJIT – Newark, US
- Hannah Bast
Universität Freiburg, DE
- Naama Ben-David
Technion – Haifa, IL
- Jon Louis Bentley
Hackettstown, US
- Rezaul Chowdhury
Stony Brook University, US
- Florina M. Ciorba
Universität Basel, CH
- Albert Cohen
Google – Paris, FR
- Alexander Conway
Cornell Tech – New York, US
- Peter C. Dillinger
Meta – Menlo Park, US
- Chen Ding
University of Rochester, US
- Clemens Grelck
Friedrich-Schiller-Universität Jena, DE
- Yan Gu
University of California – Riverside, US
- Mary W. Hall
University of Utah – Salt Lake City, US
- Bruce Hoppe
Connective Associates – Arlington, US
- Song Jiang
University of Texas at Arlington, US
- Rob Johnson
Broadcom – San Jose, US
- Timothy Kaler
MIT – Cambridge, US
- Fredrik Kjolstad
Stanford University, US
- I-Ting Angelina Lee
Washington University – St. Louis, US
- Charles E. Leiserson
MIT – Cambridge, US
- Quanquan C. Liu
Yale University – New Haven, US
- Nikolai Maas
KIT – Karlsruher Institut für Technologie, DE
- John Mellor-Crummey
Rice University – Houston, US
- William S. Moses
University of Illinois – Urbana-Champaign, US
- Prashant Pandey
Northeastern University – Boston, US
- Manuel Penschuck
Goethe University – Frankfurt am Main, DE
- Erez Petrank
Technion – Haifa, IL
- Sven-Bodo Scholz
Radboud University Nijmegen, NL
- Diane Souvaine
Tufts University – Medford, US
- Yihan Sun
University of California – Riverside, US
- Nathan Tallent
Pacific Northwest National Lab. – Richland, US
- Hans Vandierendonck
Queen's University of Belfast, GB
- David Wajc
Technion – Haifa, IL
- Jan Wassenberg
Google – Zürich, CH
- Marvin Williams
KIT – Karlsruher Institut für Technologie, DE
- Helen Xu
Georgia Institute of Technology – Atlanta, US