T. HOEFLER, M. PUESCHEL

Lecture 0: Organization

Teaching assistant: Salvatore Di Girolamo

Course Name

- **Design of Parallel and High-Performance Computing**
- **Design of Parallel and High-Performance Computing Platforms?**
- **Design of Parallel and High-Performance Computing Applications?**
- **Design of Parallel and High-Performance Computing Systems?**
- **Design of Parallel and High-Performance Computing Theory?**
- **Design of Parallel and High-Performance Computing Fundamentals?**

- **Design of Parallel and High-Performance Computing:**
  Understand principal issues involved in algorithm, software, and system development for parallel computing

The Team

- Professors: Torsten Hoefler & Markus Püschel
- TA: Salvatore di Girolamo

- Guest lecturer: maybe
- Possibly consultants for projects from Hoefler & Püschel's labs

- Course website: http://spcl.inf.ethz.ch/Teaching/2018-dphpc/

Administrative

- **Lecture: Mo 13:15 – 16:00**
- **Recitation: Do 13:15 – 15:00**
  - Takes place as announced on website
  - Sometimes used as lecture or swapped with lecture
  - Also used for project updates
- **Help:**
  - Email Salvatore: salvatore.digirolamo@inf.ethz.ch

Administrative

- **Website: http://spcl.inf.ethz.ch/Teaching/2018-dphpc/**
- **Will contain all material (slides, homeworks, schedule, etc.)**
- **Mailing list: https://spcl.inf.ethz.ch/cgi-bin/mailman/listinfo/dphpc-2018**

- **Background material:**
  - Maurice Herlihy and Nir Shavit: The Art of Multiprocessor Programming. Morgan Kaufmann, 2012
  - Papers as mentioned

Work and Grading

- **Work during semester:**
  - Regular homeworks
  - Project

- **Grade:**
  - 50% Project
  - 50% Written exam (120 minutes, in exam period as usual)

Project: Rules

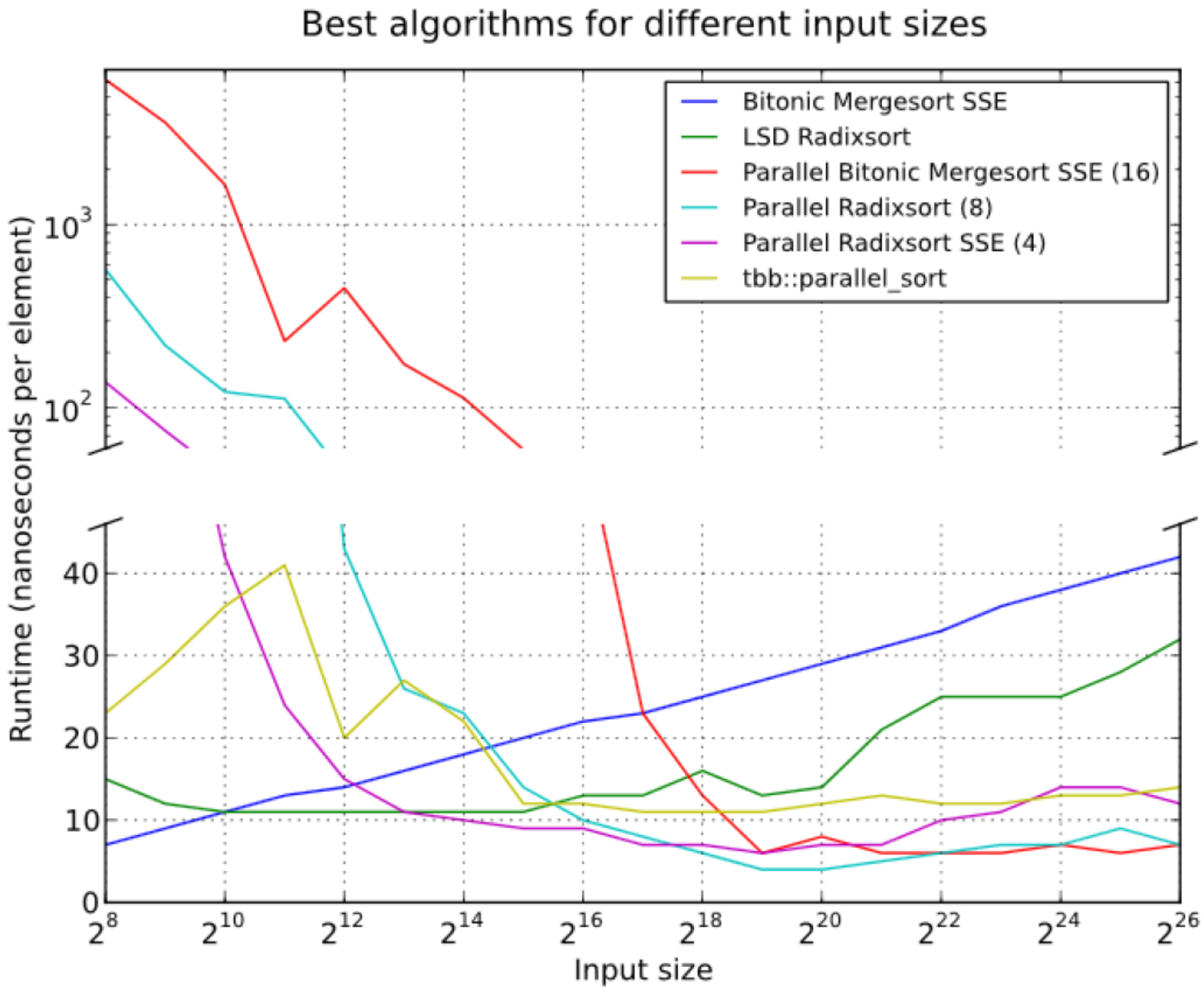
- **Count 50% of the grade (work, presentation, report)**
- **Teams of three-four**
  - Important: organize yourselves
  - You may use the mailinglist
- **Topic: Some suggestions in a minute**
- **Timeline:**
  - Oct 4th: Announce project teams to TA
  - Oct 11th: Present your project in recitation – to get a baseline
  - Oct 29th: Initial progress presentations during class
  - Last class (Dec 17th): Final project presentations
- **Report:**
  - 6 pages, template provided on webpage, due January

Projects: Performance Optimization

- **Pick an important algorithm/application**
- **Develop a parallel implementation that scales well on multicore**
- **Includes thorough benchmarking and experimental evaluation**
- **You are in charge of the project:** ***shrink or expand as necessary!***

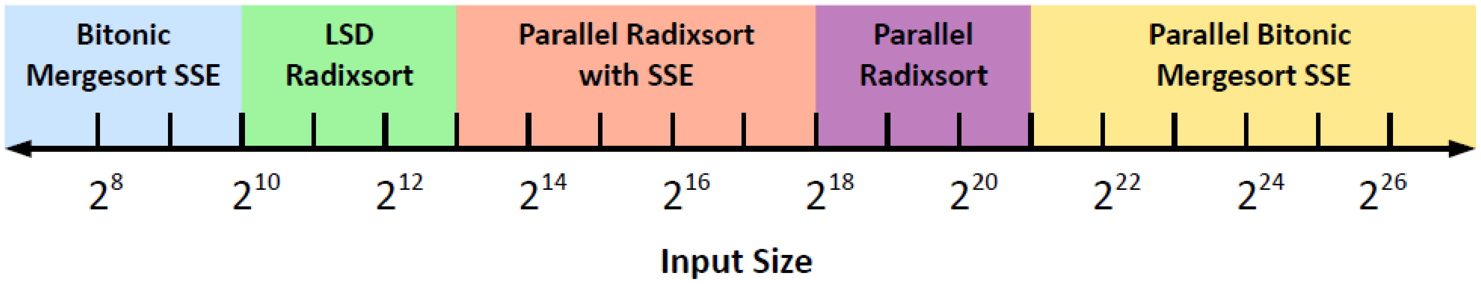
- **Requirements:**
  - No numerical algorithm (dominated by floating point operations)
    Exceptions possible if directly related to student's research
  - Not sorting or anything that is mainly sorting

Example From Before



Example From Before

- Uses our fastest implementations depending on input size and adapts #threads accordingly



Best project so far!

ACM Principles and Practice of Parallel Programming 2018

Communication-Avoiding Parallel Minimum Cuts and Connected Components

Lukas Gianinazzi
ETH Zurich
Department of Computer Science
glukas@student.ethz.ch

Pavel Kalvoda*
ETH Zurich
Department of Computer Science
kalvodap@student.ethz.ch

Alessandro De Palma
ETH Zurich
Department of Computer Science
depalmaa@student.ethz.ch

Maciej Besta
ETH Zurich
Department of Computer Science
maciej.besta@inf.ethz.ch

Torsten Hoefler
ETH Zurich
Department of Computer Science
htor@inf.ethz.ch

Abstract

We present novel scalable parallel algorithms for finding global minimum cuts and connected components, which are important and fundamental problems in graph processing. To take advantage of future massively parallel architectures, our algorithms are *communication-avoiding*: they reduce the costs of communication across the network and the cache hierarchy. The fundamental technique underlying our work is the *randomized sparsification* of a graph: removing a fraction of graph edges, deriving a solution for such a sparsified graph, and using the result to obtain a solution for the original input. We design and implement sparsification with $O(1)$ synchronization steps. Our global minimum cut algorithm decreases communication costs and computation compared to the state-of-the-art, while our connected components algorithm incurs few cache misses and synchronization steps. We validate our approach by evaluating MPI implementations of the algorithms on a petascale supercomputer. We also provide an approximate variant of the minimum cut algorithm and show that it approximates the exact solutions well while using a fraction of cores in a fraction of time.

CCS Concepts • **Theory of computation → Distributed algorithms;**

*Pavel Kalvoda was a student at ETH Zurich at the time of his involvement, but is now employed by Google Inc.

Keywords Parallel Computing, Minimum cuts, Randomized Algorithms, Graph Algorithms

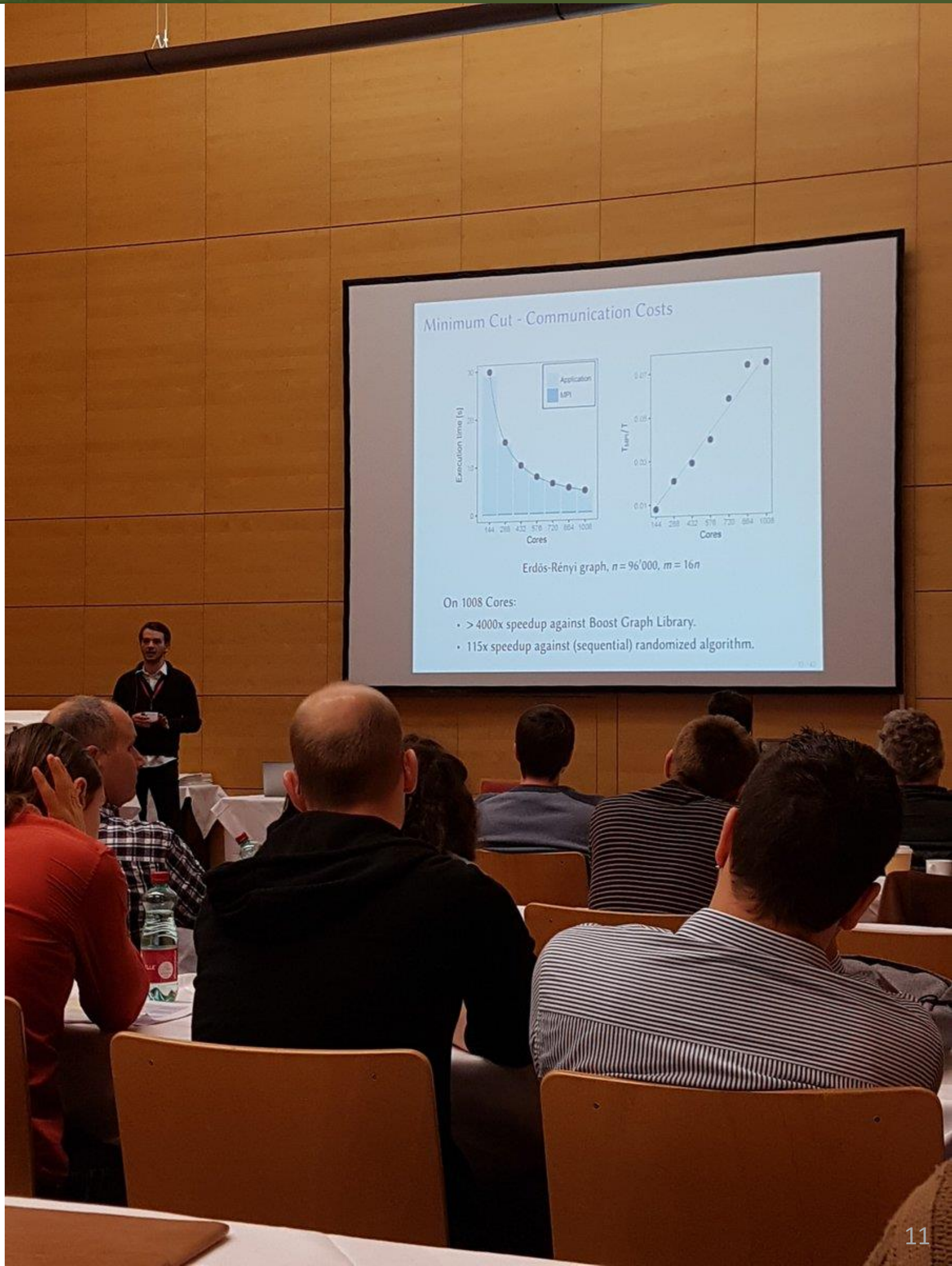
ACM Reference Format:
Lukas Gianinazzi, Pavel Kalvoda, Alessandro De Palma, Maciej Besta, and Torsten Hoefler. 2018. Communication-Avoiding Parallel Minimum Cuts and Connected Components. In *PPoPP '18: PPoPP '18: 23nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, February 24–28, 2018, Vienna, Austria.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3178487.3178504

1 Introduction

Graph computations are behind many problems in machine learning, social network analysis, and computational sciences [28]. An important and fundamental class are graph connectivity algorithms, such as finding minimum cuts or connected components.

The *global* minimum cut problem is a classic problem in graph theory; it finds a variety of applications in network reliability studies [23], combinatorial optimization [25], matrix diagonalization, memory paging, gene-expression analyses [39], and large-scale graph clustering [40]. *Connected components* is a well-studied problem with a plethora of applications, for instance in medical imaging [46], image processing [21, 32], and computer vision [49].

Designing efficient parallel graph algorithms is challenging due to their properties such as irregular and data-driven communication patterns or limited locality. These properties



Some (lame but inspiring) Project Ideas

Parallel Data Structure: Example Priority Queue

- **Modified specification:** Maintain a collection of data items, identified by a key. Finding the k smallest items (with the k smallest keys) should be supported in O(k) time. Finding any item by key should also be supported.

Required Operations

- **queue_t init()**
- **void insert(queue_t q, void* data, uint64_t key)**
- **void*find(queue_t q, uint64_t key)**
- **void delete(queue_t q, uint64_t key)**
- **void*pop_front(queue_t q, int k) // returns k smallest elements**
- **void finalize(queue_t q)**

Parallel Priority Queue (II)

- **Requirements contd.**
  - Multiple threads will be accessing the queue simultaneously (with all operations)
  - Code may be written in C/C++ (gcc inline assembly is allowed ;-))

- **Tips:**
  - Experiment with different locking strategies and compare the performance
  - Pay attention to larger number of threads
  - Maybe try MPI-3 One Sided

To make is more interesting: Brodal et al.: “A Parallel Priority Queue with Constant Time Operations”, JPDC’98
Check parallel in-time simulations from computational science for use-cases!

Collective Communications

- **Assume P threads in shared memory**
- **Each thread p has:**
  - a set of input elements $i_{j,p}$ ($0 \leq j < n-1$)
  - a set of output elements $o_{j,p}$ ($0 \leq j < n-1$)

- **The post-condition (result) is:**
  - $o_{j,p} = \sum_{p=1}^{P} i_{j,p} (0 \leq j < n)$

  - i.e., all $o_{j,p}$ are identical on all p

- **Tips:**
  - Use the memory hierarchy and CC protocols (inline assembly is allowed!)
  - First optimize small n, then large n

Check: Li et al.: "NUMA-Aware Shared Memory Collective Communication for MPI", HPDC'13

Parallel Algorithms: Example BFS

- **Generate an** Erdős–Rényi **graph G(n,p) given n and p**

- **Perform a breadth-first search (BFS) from n/2 vertices**
  - Print the average maximum distance for any vertex

- **Your implementation should exploit all available cores and perform the BFS as fast as possible**

Check: Lin et al.: “ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds”, SC18

Parallel Graph Algorithms

- **Many more!**
  - Connected Components (CC)
  - Single-source shortest path (SSSP)
  - All-pairs-shortest path (APSP) - too simple, looks like MatVec
  - Minimum spanning tree (MST)
  - Vertex coloring
  - Strongly connected components
  - … pick one and enjoy!
- **Others**
  - A* search
  - Various ML and AI algorithms (only nontrivial ones)

- ***Always implement infrastructure to validate your code!***

Check: Quinn, Deo: “Parallel graph algorithms”, CSUR’84 (outdated but still good base) – HUGE space to invent!

Mind the Lecture!!!

- **Try to relate your project to the contents of the lecture!**
  - E.g., analyze sequential consistency (was very successful!)
  - E.g., deal with memory models!
  - E.g., write litmus tests for various architectures (would be very cool)
  - Analyze overheads of atomic operations on various architectures in detail
  - Reason about the performance obtained
  - Many more (be creative!)
  - Or talk to the TA(s)

- **Remember: you have until the end of October**
  - You can also check the slides from last year for later lecture topics (mind that this year will be slightly different!)
  - This is of course all up to you