Design of Parallel and High-Performance Computing
Fall 2019

*Lecture:* Organization

**Instructor:** Tal Ben-Nun & Markus Püschel
**TA:** Timo Schneider

---

**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:
  *Understand principal issues involved in software and system development for parallel computing*
The Team

- Professors: Tal Ben-Nun & Markus Püschel
- TA: Timo Schneider & Others
- Guest lecturer: maybe
- Possibly consultants for projects

 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
  - Room was changed to CHN C 14
- Help:
  - Email Timo: timo.schneider@inf.ethz.ch
Administrative

- Will contain all material (slides, homeworks, schedule, etc.)

- 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

- Each project is done in teams of four.
- You can use the mailing-list dphpc-forum-2019 (everybody is subscribed initially) to find project partners, offer a project etc.
- Once you have a team (even without project) email the TA and we will take you off the list
- Ideas for projects: see below. Projects have to be approved by the TAs/lecturers (see next slide)
- We will track progress during the semester

Project: Timeline

- **Before October 11:** Find team (let TA know names). Find project. If you have a suggestion send email with topic and rough plan and references to TAs and lecturer for approval. Note that this may take more than one iteration.
- **October 11:** You have a team and an approved project.
- **During semester:** We will check progress in some way. Procedure and possible dates to come.
- **End of semester:** Project presentations during lecture/recitation hours.
- **January 17:** Project reports due (6 pages, conference style, information on web).
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

![Chart showing runtime vs. input size for different sorting algorithms](chart.png)
Example From Before

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

![Diagram showing input size vs algorithms]

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 continued**
  - 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
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 \( n \)

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
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!

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!
  - Reason about the performance obtained
  - Many more (be creative!)
  - Or talk to TA

- Remember: you have until the October 11th
  - You can also check the slides from last year for later lecture topics
  - This is of course all up to you