Design of Parallel and High-Performance Computing
Fall 2017
Lecture: Organization of the Course

Instructor: Torsten Hoefler & Markus Püschel
TA: 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: 
  Understand principal issues involved in software and system development for parallel computing
The Team

- Professors: Torsten Höfler & Markus Püschel
- TA: Salvatore di Girolamo
- Guest lecturer: maybe
- Possibly consultants for projects

- Course website: http://spcl.inf.ethz.ch/Teaching/2017-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/2017-dphpc/](http://spcl.inf.ethz.ch/Teaching/2017-dphpc/)
- Will contain all material (slides, homeworks, schedule, etc.)

- Background material:
  - 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
  - Important: organize yourselves
  - You may use the mailinglist

- Topic: Some suggestions in a minute

- Timeline:
  - Oct 12th: Announce project teams to TA
  - Oct 19th: Present your project in recitation
  - Nov 6th: Initial progress presentations during class
  - Last class (Dec 18th): 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

Best algorithms for different input sizes

- Bitonic Mergesort SSE
- LSD Radixsort
- Parallel Bitonic Mergesort SSE (16)
- Parallel Radixsort (8)
- Parallel Radixsort SSE (4)
- tbb::parallel_sort

Run time (nanoseconds per element) vs. Input size.
Example From Before

- Uses our fastest implementations depending on input size and adapts #threads accordingly.
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
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!
  - E.g., write litmus tests for Xeon Phi (would be very cool)
  - Analyze overheads of atomic operations on Xeon Phi in detail
  - Reason about the performance obtained
  - Many more (be creative!)
  - Or talk to TA

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