

Design of Parallel and High Performance Computing

Fall 2013

About projects

Instructors: Torsten Hoefler & Markus Püschel

TAs: Timo Schneider, Arnamoy Bhattacharyya



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Project: Rules

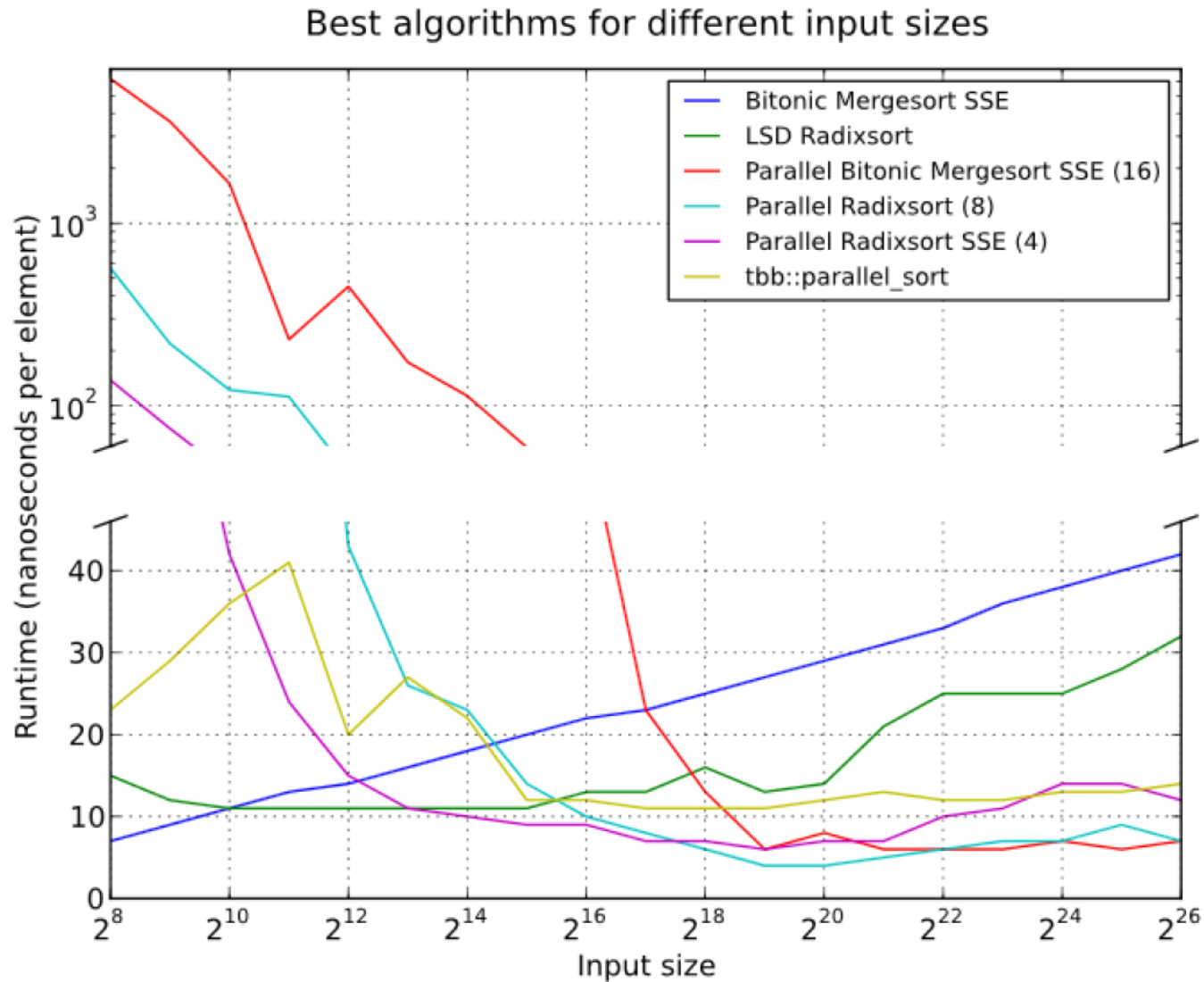
- **Count 50% of the grade (work, presentation, report)**
- **Teams of three**
 - Important: organize yourselves
 - You may use the mailinglist
- **Topic: Some suggestions in a minute**
- **Timeline:**
 - Mid Oct: Announce project teams to TAs
 - End Oct: Present your project in recitations
 - Late Nov/early Dec: Possibly progress presentations
 - Last week of class: Final project presentations
- **Report:**
 - 6 pages, template provided, 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

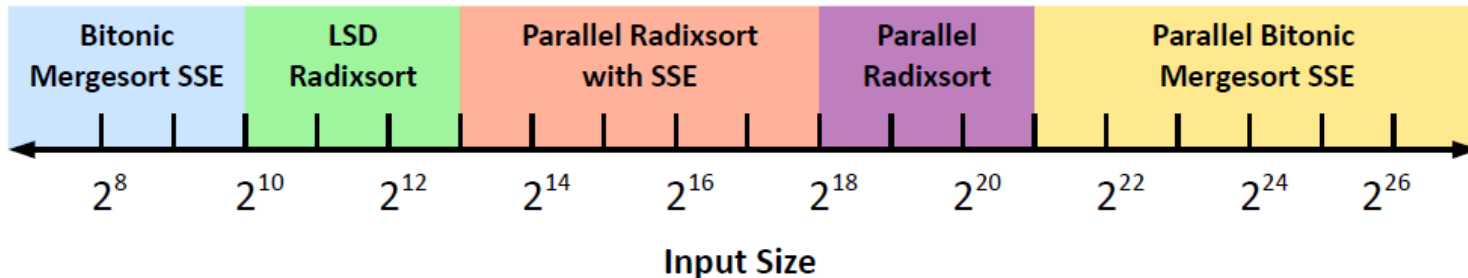
- **Requirements:**
 - No numerical algorithm (dominated by floating point operations)
Exceptions possible if directly related to student's research
 - Not sorting or anything that is mainly sorting

Example From Before



Example From Before

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



Project Proposals

Advisor: Torsten Hoefler

TA: Timo Schneider

Parallel Priority Queue (I)

- **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:
 - $$o_{j,p} = \sum_{p=1}^P i_{j,p} \quad (0 \leq j < n)$$
 - 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 BFS

- **Generate an ER graph $G(n,p)$ given n and p**
- **Perform a breath first search 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)
- SSSP
- APSP (maybe 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)

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 very cool)
 - Analyze overheads of atomic operations on Xeon Phi in detail
 - Maybe even write a checking tool?
 - Many many more (be creative!)
 - Or talk to the Tas/Assistants

- **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

Schedule

- **Some recitations will be used to demonstrate concepts in practice**
 - E.g., OpenMP basics, MPI basics, ...
- **We will discuss “how to measure and report performance”**
 - This is a complex topic often done wrong