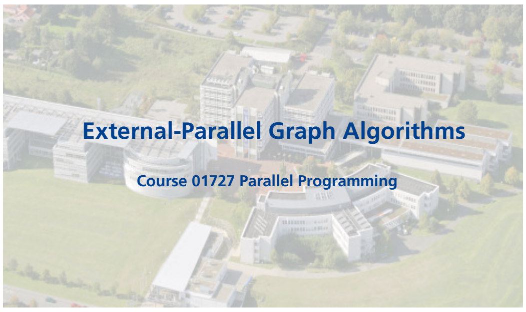


Parallelism and VLSI Group
Prof. Dr. Jörg Keller

Overview

- External-Memory Algorithms
- Parallel Graph Algorithms
- Application
- Algorithms
- Summary

External-Memory Algorithms

- External-memory alg = data set too large to hold in mem
- Data on hard disk:
  bandwidth ok for few access patterns
  pagewise access
  latency high
  random access very slow
- Best known example:
  external sorting (mostly merge sort, mostly databases)

Parallel Graph Algorithms I

- Graph representation:
  - explicit by adjacency list
  - implicit by oracle
- Oracle: call black-box code with node x as parameter
  obtain f(x) such that (x,f(x)) is edge
- Advantage of explicit representation:
  graph can be changed, e.g. pointer jumping
- Advantage of implicit representation:
  able to handle really large graphs without ext. memory

Parallel Graph Algorithms II

- Example:
  given digraph G with outdegree 1 by oracle
  compute strongly connected components (SCCs)
- Two variants:
  - indegree also exactly 1 (know function f is bijective)
  - indegree may vary (know nothing about function f)

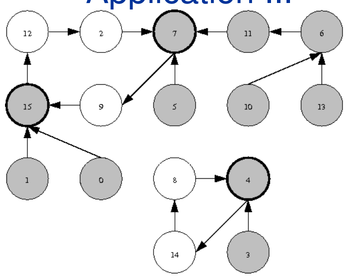
Application I

- Consider finite state machine accepting no input
  (except in init phase)
- State graph has outdegree exactly 1
- Examples:
  pseudo-random number generator
  stream cipher
  iterated block cipher
  cryptographic hash chain

Application II

- Oracle = code of state-transition function
- State space typically large: 2^{64} at least
- Strongly connected components = cycles
  cycle lengths important parameter wrt security

Application III



Algorithms I

- List ranking as first excercise
- Given: List of length n in array succ[1:n]
  end of list points to itself
- Wanted: for each list element, distance to end of list
- For(t=1;t<log n;t++)
  do in parallel for i=1..n {
  dist[i] += dist[succ[i]]; succ[i] = succ[succ[i]];
  }

Algorithms II

- Pointer doubling in external memory:
- Store list of edges (i,succ[i]) twice on external memory
- Sort one copy with i, one copy with succ[i]
- For each i:
  find predecessor j in 2nd copy, succ[j] = i (possibly mult.)
  find successor succ[i] in 1st copy
  write (j,succ[i]) to disc
- Iterate log n times: $O(n/p*(\log(n))^2)$

Algorithms III

- Find SCCs of deg1-graph given by oracle f
  if each SCC contains a node with property A
- For(all nodes x with property A){
  follow path x, f(x), f(f(x))… till reach node y with A
  write (x,y,$dist_{xy}$) to disk
  }
  do{
  make copy of edge list with reversed edges;
  sort 1st copy with x, sort 2nd copy with y;
  for each (x,y) in 1st copy:
  write (x,y) to disk if (x,y') in 2nd copy
  }while(edges have been removed);

Algorithms IV

- Algorithm „shaves" leaves from graph
- May iterate for long, expect distance to cycle O(sqrt(n))
- Therefore: use either if known that graph is shallow
  or use until size reduced that fits into main mem

Algorithms V

- Find most cycles (=SCCs) for deg1-graph with bij. oracle f
- Preprocessing:
  define as many random anchor nodes as fit on your disks
  for(each anchor x){
    follow path x, f(x), f(f(x)),… until reach other anchor y
    write to disk (x,y,$dist_{xy}$)
  }
- Converts graph with oracle into smaller graph with adjacency list
  contains only cycles with $\geq$1 anchor

Algorithms VI

- How to detect that node is anchor?

  If assumed that deg1-graph is random:
  use deterministic anchors e.g. all with 24 zero bits

  If not: use additionally pseudorandom permutation pi
  where pi(x) and $pi^{-1}(x)$ can be computed fast
- Why so many anchors that disks are needed?
  Chance of putting anchor onto small cycle increases
  Find more cycles!

Algorithms VII

- Compute cycles in anchor graph in several rounds
- Each round:
  load edges (x,y) from largest x, as fit into main mem
  compact all edges with (y,…) also in main mem
- Remove references to those edges in edges on disk
- Parallel time: $O((n/m)^2/p)$

Summary

- Ext-mem algorithms gain importance:
  large data sets, complex memory hierarchies
- Many ext-mem algorithms use sorting
- Currently: some better algorithms for undirected graphs
- Algorithms very complex