Today I will be talking about an empirical

A Comparison of Parallel Algorithms for Connected Components

John Greiner
School of Computer Science
Carnegie Mellon University
jdg@cs.cmu.edu

Previous Theoretical Work

This study only looks at Concurrent Read Concurrent Write algs.

Previous CRCW work includes the algorithms in this table

where the highlighted algs. are those used for this study.

Won't talk about SV due to its similarity with AS. The AS, RM algs. were chosen largely for their simplicity, whereas the CV and Gazit algs. much more complicated, probably not practical for sizes of graphs tested

HY and an optimized AS alg. were developed as part of this work, and will describe HY later.

Of the algs. used, only RM is randomized.

Extra log n time factor for RM, HY due to use of graph contraction. In practice, having algs. w/reasonable work & time, & low constant factors is more important than optimality.

Note that RM, HY have a lower worst-case work bound for some classes of graphs.

Previous Pragmatic Work

While much theoretical work, **Little general-purpose empirical work**

- **Physics and vision — most algorithms specialized for grids**
- Studies of general-purpose algs. typically tested only a single alg. on single machine.

So, unlike previous work, this is an

Empirical comparison of multiple general-purpose algorithms

- **Coded in NESL — Provides hardware-independent PRAM + scan model**
- **Tested with multiple classes of graphs**
- **Executed on multiple architectures** with goal that results generalize to other hardware. To this end, used hardware with distinct characteristics, incl. different relative comm./comp. costs. In particular, 8K proc. CM-2; 1 proc. of Cray Y-MP/C90, using vector facilities as 128 virtual procs. Newer work with Marco Zagha uses multiple procs. of Cray.

Hybrid as fast as previous algorithms specialized for grids

- **On 150 x 150 x 150 grid, 40% of edges:**
- **Fink-Baden-Jansen ’94 on 256 proc. Paragon: 0.9 sec.**
- **Hybrid on Cray: 1.6 sec. — New: 0.54 sec. (1 proc.), 0.15 sec. (4 proc.)**

Classes of Graphs Used in Testing

Grids, 2D & 3D

Commonly used in practice

30% or 60% of 2D edges, 20% or 40% of 3D edges

Random — 2% of edges from complete graph

Large graphs increasingly dense

since fixed percentage of edges is used, and number of edges grows quadratically

Tertiary — each node picks 3 targets for edges

Somewhat intermediate case

Empirically, Size of separators affects rate of contraction and thus speed of algs.

Grids have small separators, while others have large separators. Paper typo for random.

Comparison of Algorithms

Before I describe the algs, let’s look at how they perform on these classes of graphs.

Times (sec.) on 8K proc. CM-2, graphs of 500,000 edges

HY consistently fastest, 3-8 times faster than AS’. RM 2-5 times faster than AS’.

Advantage of RM, HY is most pronounced on highly connected graphs

(2D30: 10.7, 4.4, 3.8; 3D40: 15.0, 6.9, 4.5; Rand: 12.1, 8.1, 3.4; Tert: 58.5, 11.0, 7.3)

(AS about 5% faster than SV)

While not shown, optimized AS is about 3 times faster than the original (factor of 2 due to eliminating dummy nodes/edges)

Algs. slowest on tertiary graphs, but the times on random graphs are comparable to those of grids (on this size of graph). Later, these comparisons will be related to how these graphs contract.

Note that the similar comparison in paper uses the wrong size of random graphs. Correct times for random graphs are approximately double of those shown in Figure 1 of the paper.

Some Ideas in CC Algorithms

Some key ideas permeate the various parallel CC algs., incl. the following. Based on building

Trees of nodes — known to be in same CC — roots as representatives of trees

Shortcutting — halves height of trees - used in AS, HY, useful in conjunction with

Hooking — combines pairs of trees — Must avoid creating cycles.

Hooking with subsets of edges is key for Gazit's optimality

Contracting — removes nodes and/or edges shrinks graph for later iters. RM, HY

This diagram is an example of one pair of trees being hooked & the result being shortcutted.

Red arrows are parent pointers of trees, black arrow is the graph edge used for hooking. While edges are undirected, they are given an orientation for purposes of hooking. An edge is used for hooking only if its source node is of depth at most 1. Thus, hooking works best with trees that have been shortcutted. Hooking & shortcutting don't affect graph, only trees. Contraction affects graphs and would remove edge used in hooking.

Now talk about hybrid alg., developed as part of this work, and uses some of these ideas.

Hybrid Algorithm

HY uses contraction ph. to find the components & expansion ph. to propagate their names. Now illustrate one iteration of contraction.

1) Graph and initial trees, where each node is its own parents. Red arrows are parent pointers, red numbers are parent numbers.

2) Hook to lower numbered neighbors. When there are multiple such neighbors, we pick one arbitrarily. Alternate between using lower and higher neighbors to avoid worst-case of O(n) iters.

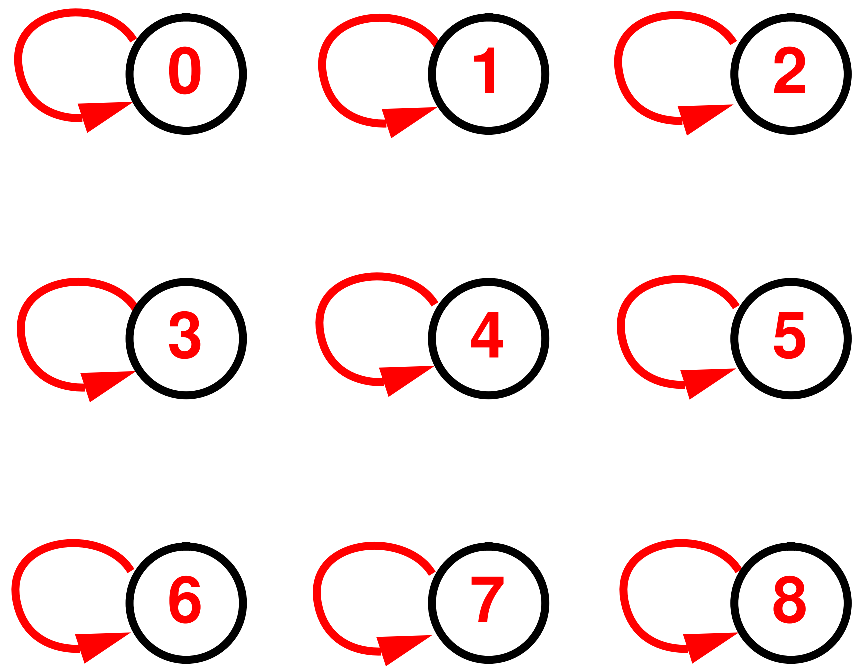
3) Shortcut until all trees of depth at most 1. Most of the alg.'s work is this repeated shortcutting. Each shortcut consists only of 1 read operation. This completes our modification of the trees.

4) Rename edges' endpoints with their parents. May result in duplicate and self-edges.

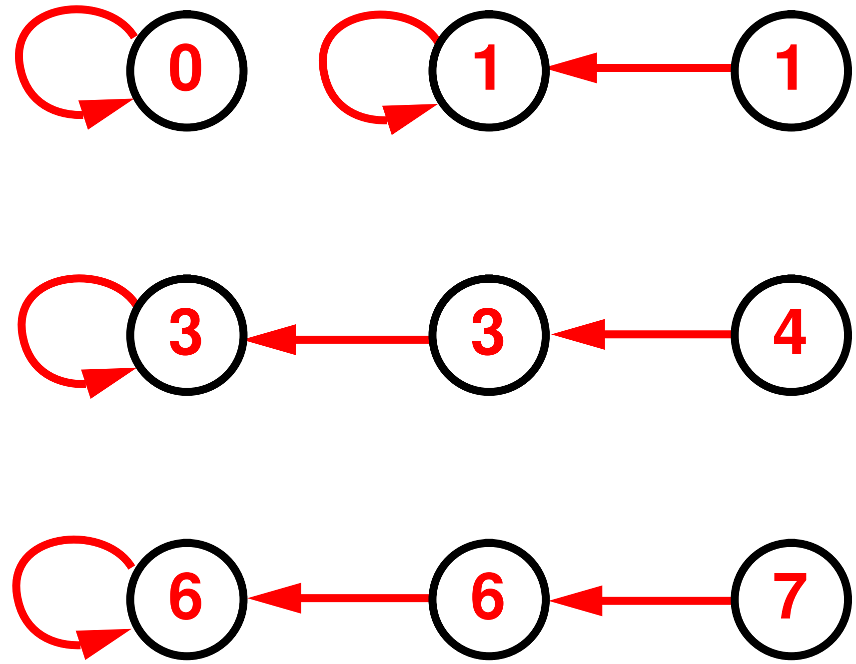
5) Remove self-edges, non-root nodes, singleton nodes. Remaining edges are those which had linked two different trees.

6) Renumber nodes. Resulting graph is used for next iteration. Contraction phase terminates if no edges left.

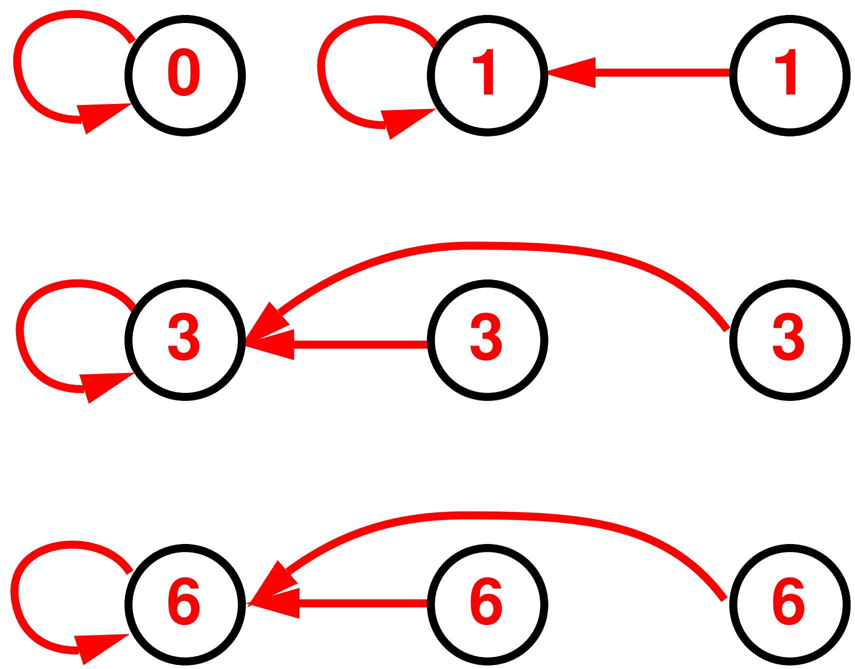
Expansion phase propagates names backwards along the parent pointers.



Initial trees

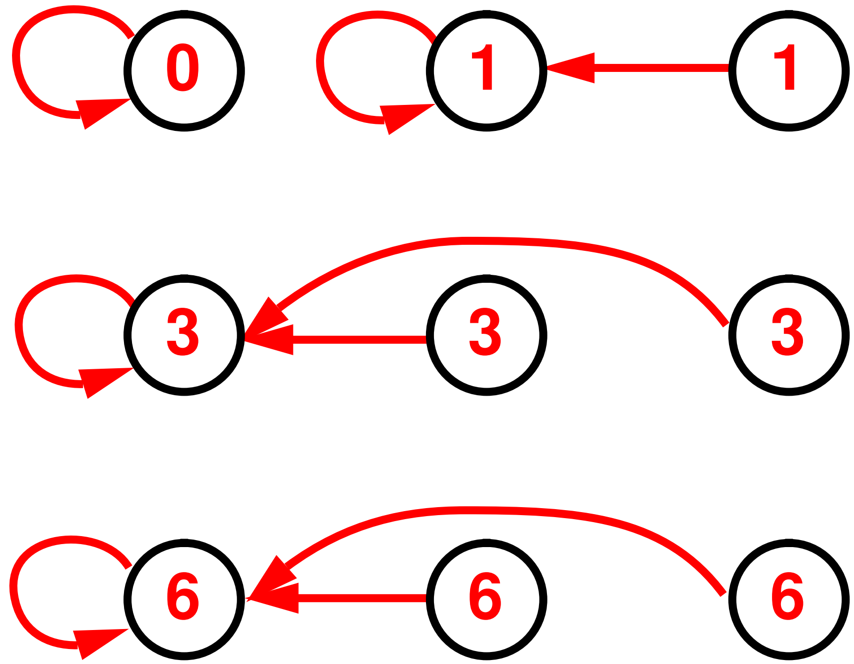


Trees after hooking

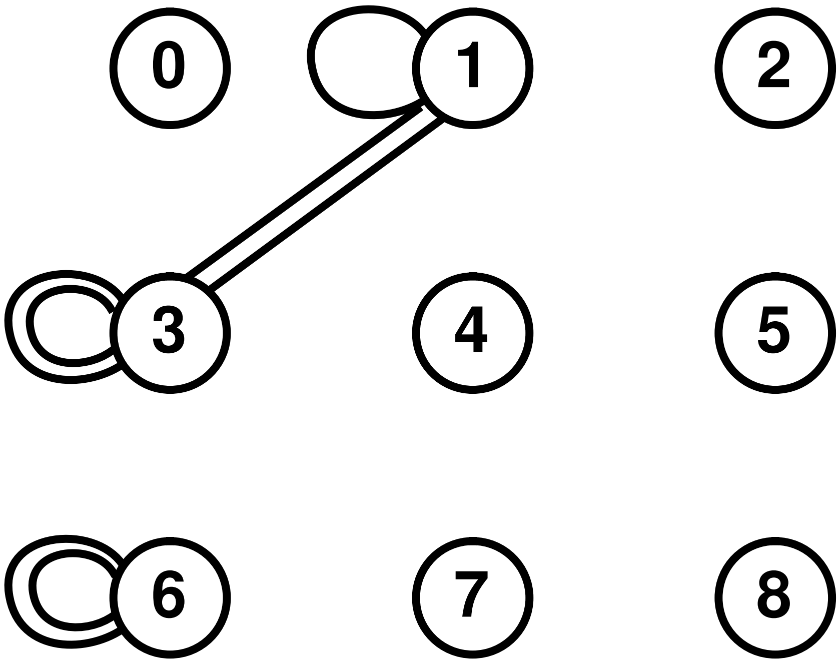


Trees after shortcuting

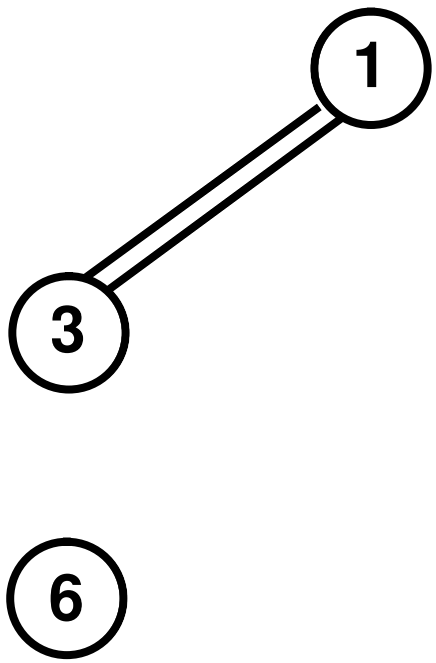
Hybrid Algorithm



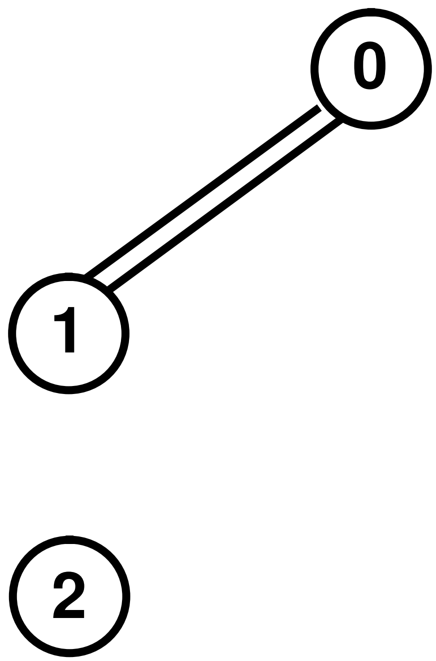
Trees after shortcuting



Graph after renaming edges with parents



Graph after contracting



Graph after renumbering

Hybrid Algorithm in Theory and Practice

Theory:

Non-optimal, O((m+n) log n) work, O($\log^2$ n) time

O(log n) iterations since number of nodes decreases geometrically (at least half every other iter) each of O(m+n) work, O(log n) time (dominated by contraction, short-cutting)

Practice:

Low constants each shortcut is only 1 read, and it uses...

Simple hooking step esp. compared to AS, which uses 2 much more involved forms of hooking

Extensive shortcutting benefits hooking and contraction as edges are used in hooking on earlier iterations & graphs contract faster.

Quick contraction — typically removes about 80% of nodes each iteration

Work-efficient for planar graphs — contraction

Rate of Edge Contraction

This plot shows the rate of edge contraction in HY. Plot for RM similar.

Amount of work done is proportional to the sum of # of edges remaining on each iteration.

Alg. clearly requires more work on tertiary graphs. However, at least for the sizes tested, it requires comparable work on grids (which contract quickly, but require many iterations) and on random graphs (which contract slowly initially, but require few iterations). This echoes the timing results shown previously.

O(n) nodes are contracted every other iteration, so at least O(n) edges are also contracted every other iteration.

This means that dense graphs contract slowly, becoming increasingly dense until almost fully connected. At this point, the # of edges decreases quickly as it is (essentially) bounded by the square of remaining nodes.

# of edges in grids decreases geometrically since grids have O(n) edges & planar graphs contract to planar graphs.

Concluding Remarks

Hybrid algorithm empirically fastest <u>of those tested</u>

- Combines graph contraction and extensive shortcutting
- Low constants
- Emphasizes many simple steps, rather than few complicated steps
- As fast as previous algorithms specialized for grids

- On Cray, grids of 2-4M nodes processed in about 1 second
- 3.5-10.8 times faster than serial DFS on 133 MHz DEC Alpha

<u>Newer results, using Fortran on Cray, with improved shortcutting routine: 5-11M nodes/sec on 1 proc. (~2.5 times faster), upto 65M nodes/sec on 8 procs.</u>

Comparative results very similar on distinct hardware

- Results should generalize

<u>Differences due to relative time needed for communication vs. computation, E.g., shortcutting, and thus HY alg., is relatively better on Cray where comm. as fast as comp.</u>

Hybrid Algorithm on Cray Y-MP/C90 (1 Proc.)

Very consistent times when compared by edges

Inefficient RM generation routine didn't allow large graphs to be tested.

Rate of Node Contraction

Some details about the classes of graphs

Class	Avg. Degree	Separator Size	Diameter
2D: 30%, 60%	1.2, 2.4	$O(n^{1/2})$	$O(n^{1/2})$
3D: 20%, 40%	1.2, 2.4	$O(n^{2/3})$	$O(n^{1/3})$
Tertiary	6	O(n)	O(log n)
Random: 2%	.04 (n-1)	$O(n^2)$	$O(.02\ n^2)$

Hybrid code

```
function flip_edges(edges,flips) = {(select(flip,to,from),select(flip,from,to)) : (from,to) in edges; flip in flips};
function flip_up(edges) = flip_edges(edges,{from > to : (from,to) in edges});
function flip_down(edges) = flip_edges(edges,{from < to : (from,to) in edges});
function flip_all(edges) = {(to,from) : (from,to) in edges};

function parents_edges(parents,edges) =
{(pfrom,pto) : pfrom in parents -> edges_froms(edges); pto in parents -> edges_tos(edges)};

function shrink_edges(parents,edges) =
{(pfrom,pto) : (pfrom,pto) in parents_edges(parents,edges) | pfrom /= pto};

function shortcut(parents) = parents -> parents;

function shortcut_n(parents,n) = if n <= 0 then parents else shortcut_n(shortcut(parents),n-1);

function shortcut_max(parents) =
let gparents = shortcut_n(parents,3); ggparents = shortcut(gparents);
in if all({gp = ggp : gp in gparents; ggp in ggparents}) then ggparents else shortcut_max(ggparents);
```

Hybrid code

```
function compress(newnodes,oldnodes,edges,iter) =
let flags = {oldn == newn and -1 /= c: oldn in oldnodes; newn in newnodes;
                                              c in (dist(-1,#oldnodes) <- edges) <- flip_all(edges)};
    flip_es = if (iter and 1) == 0 then flip_down(edges) else flip_up(edges);
    edges1 = parents_edges(enumerate(flags),flip_es);
in (pack_index(flags),edges1);

function hybrid_alg(nodes,edges,iter) =
if zerop(#edges) then nodes
else let nodes1 = shortcut_max(nodes <- edges);
          edges1 = shrink_edges(nodes1,edges);
          (newnodes,newedges) = compress(nodes1,nodes,edges1,iter);
          roots = hybrid_alg(index(#newnodes),newedges,iter+1);
      in shortcut(nodes1 <- {(i,v) : i in newnodes; v in newnodes -> roots});

function cc_hybrid(edges,numnodes) = hybrid_alg(index(numnodes),flip_up(edges),0);
```