A Parallel Implementation of the Push-Relabel Max-Flow Algorithm with Heuristics

6.884 Final Project, Spring 2010

Victoria Popic, Javier Velez

Background

- Applications

  resource allocation, scheduling, linear programming problems, graph problems (max bipartite matching)

- Algorithms

  - augmenting paths (Ford and Fulkerson, Edmonds-Karp, Dinitz)

  - preflow-push (Goldberg and Tarjan) – best in practice Goldberg's push-relabel *hipr* algorithm

Max-Flow Push-Relabel Algorithm

- G = (V, E), s, t; c(u, v); f(u, v); |f|
- *preflow:* allow excess flow at a vertex
- assign a distance from sink value to each vertex; d(s) = |V|, d(t) = 0

PUSH(u,v)

// Applicability: u is active, $c_f(u,v) > 0$ and $d(u) < d(v)$
$\delta = min(e(u), c_f(u,v))$
$f(u,v) \leftarrow f(u,v) + \delta$
$f(v,u) \leftarrow f(v,u) - \delta$
$excess(u) = excess(u) - \delta$
$excess(v) = excess(v) + \delta$

RELABEL(u)

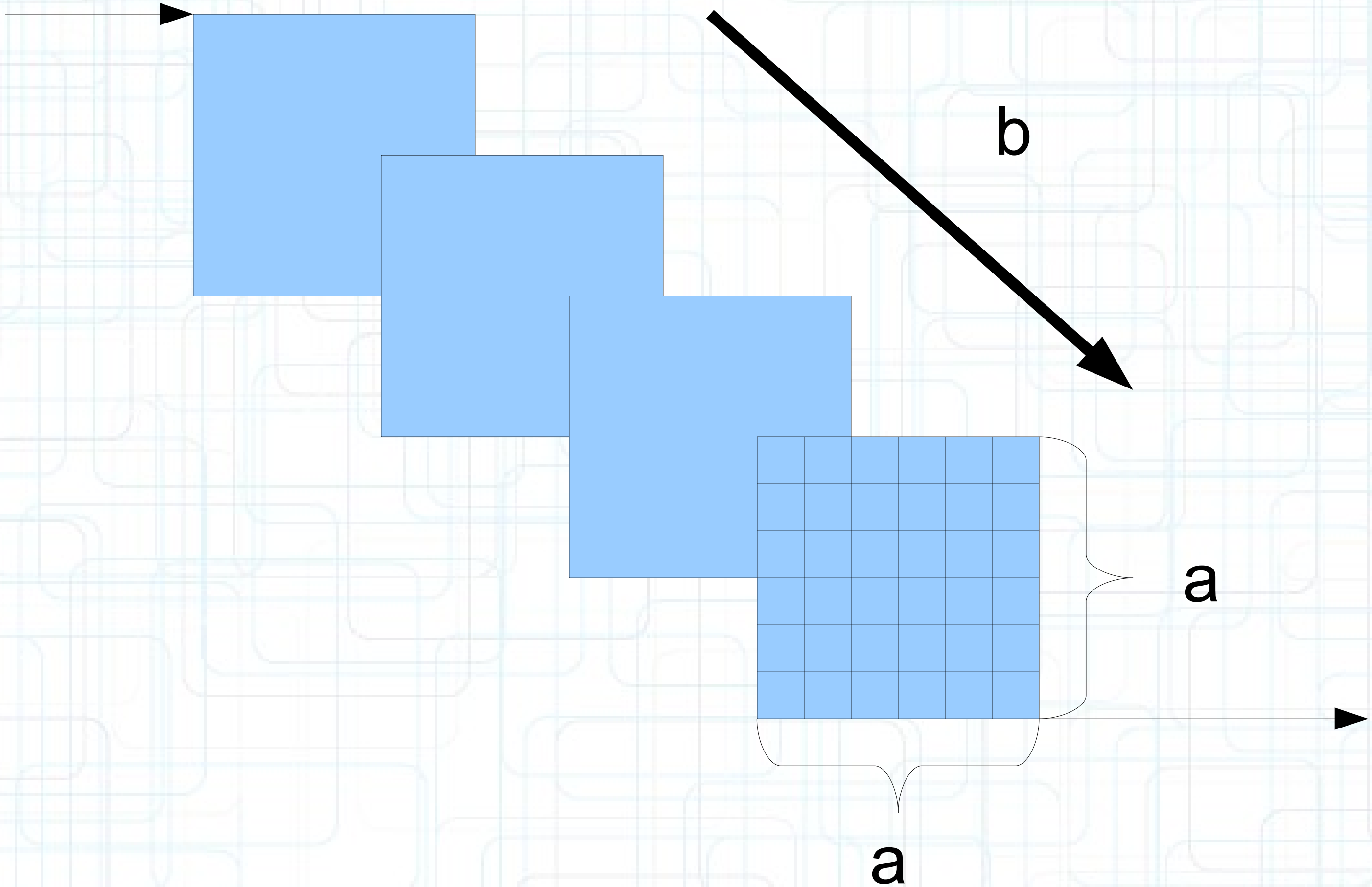
// Applicability: u is active,
// $\forall w \in V, if c_f(u,w) > 0 \Rightarrow d(u) \leq d(w)$
$d(u) \leftarrow min(d(w) | (u,w) \in E_f) + 1$

DISCHARGE(u)

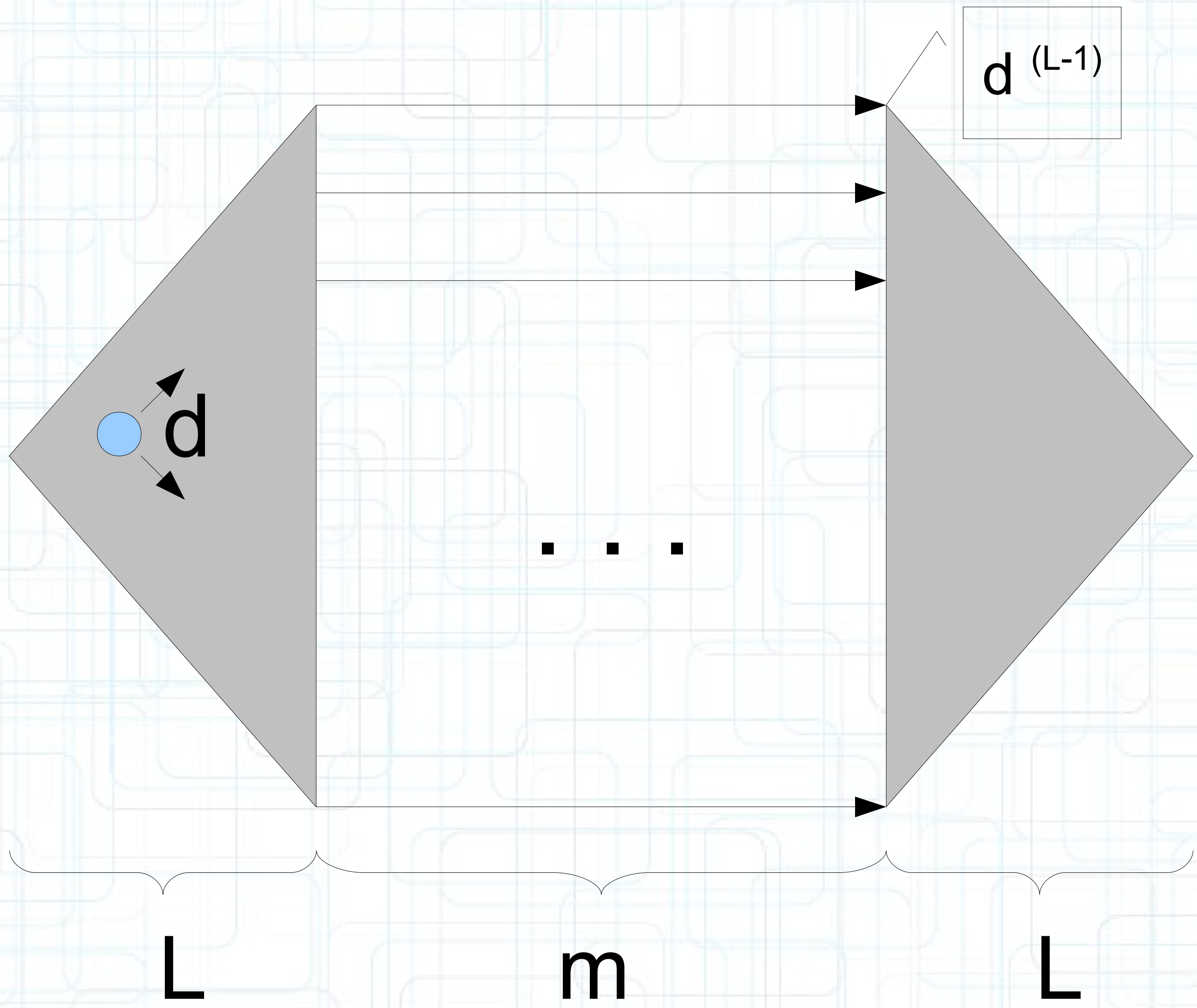
// Applicability: u is active
while $excess(u) \neq 0$
PUSH or RELABEL (u)

- ordering for discharge: FIFO / LIFO; highest distance nodes first (best)

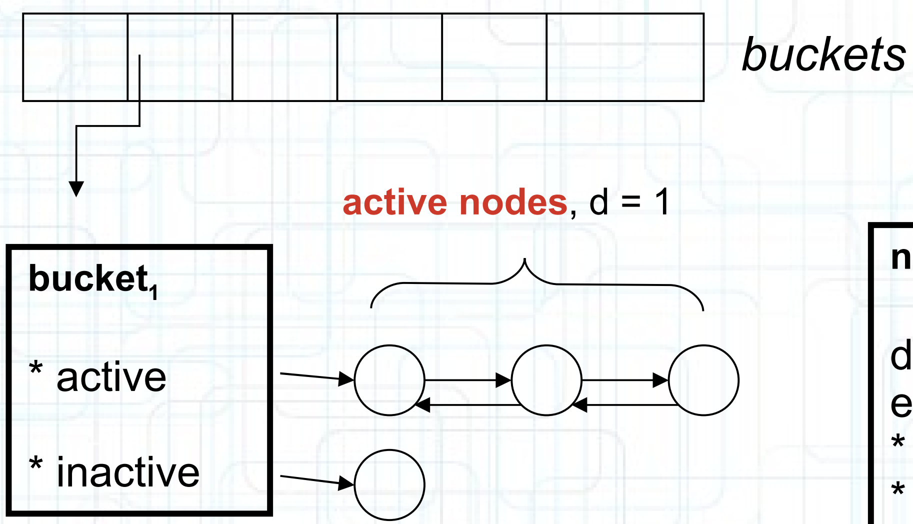
RMF Graphs Parametrized by (a,b)



Trees Parametrized by (L, d, m)



HI-PR (Goldberg) Data Structures



node

d
excess
* prevNode
* nextNode

bucket

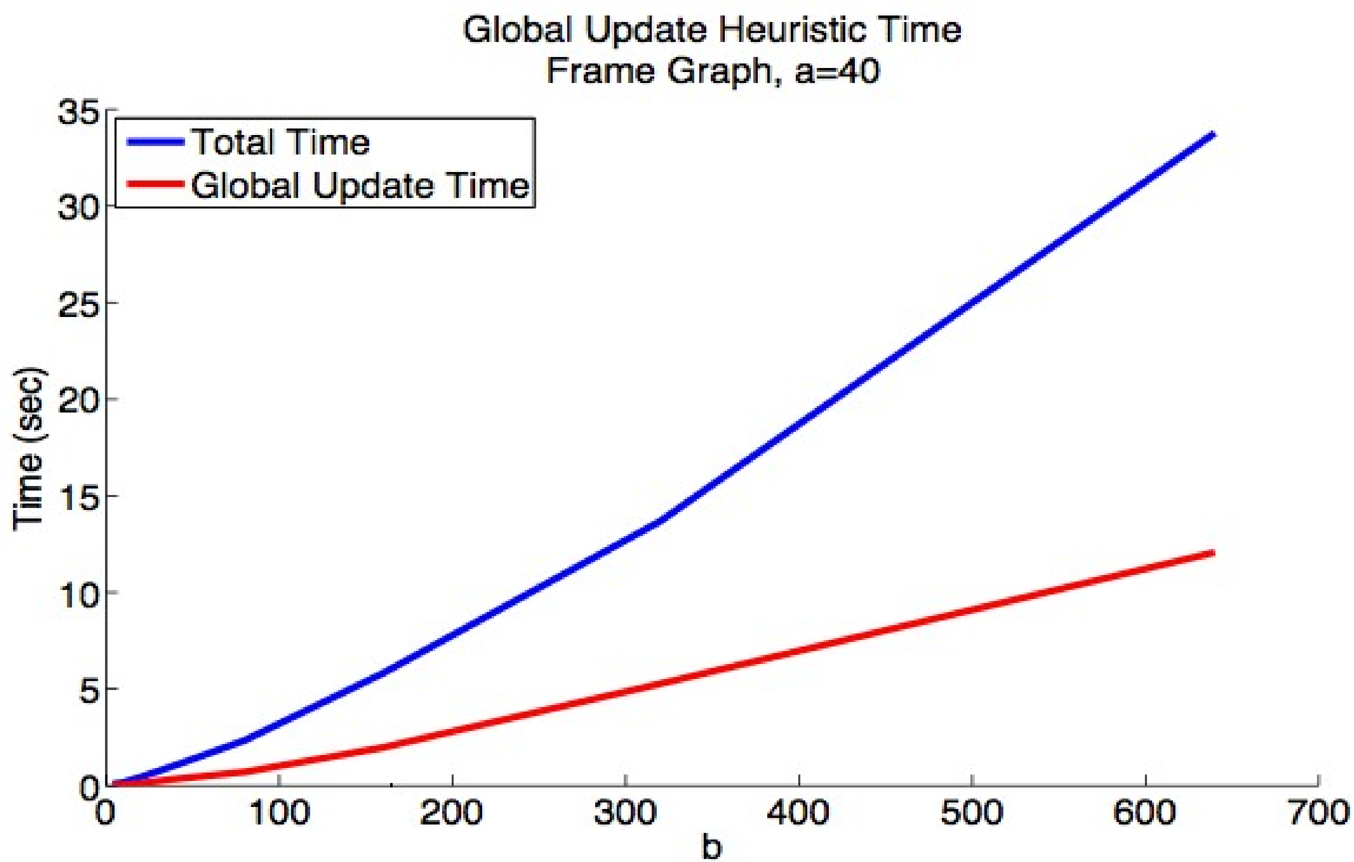
* active
* inactive

Global Relabeling Heuristic

- ***backwards* BFS** from sink: computes exact distances of nodes from the sink
- updates buckets and node data (distance and current arc)

```
for each (node i : inactive and active list of bucket k)
         for all neighbors j s.t. (j, i) is an admissible arc
                  update j: j.d = k+1, j.current = j.first
                  if(j.excess > 0)
                      add j to (k+1) bucket's active list
                  else
                      add j to (k+1) bucket's inactive list
```

Global Heuristic Time



Parallel Global Relabeling Heuristic with Pennants and Bags

- use ***Bag reducers*** to store the nodes in the buckets during search (4 Bag reducers for 2 levels of active and inactive lists)
- after we're done computing layer *k*, *set the pointers of bucket k* to the nodes in the active and inactive reduced bag
- we need to maintain a ***node chain inside*** our bags
  - modify bag's **INSERT**(node) and **MERGE**(bag) to maintain pointers between all the nodes inside the bag
- race: when checking if a node has been visited already, use atomics/locks to avoid duplicates in the buckets

Parallel Global Relabeling Results

- *rmf* graph (*a=100, b=100*) |V| = 1,000,000, |E| = 4,950,000
- global update time: serial = *7.848 (s),* parallel = *3.932 (s)*

speedup = 2

Cilkview Results

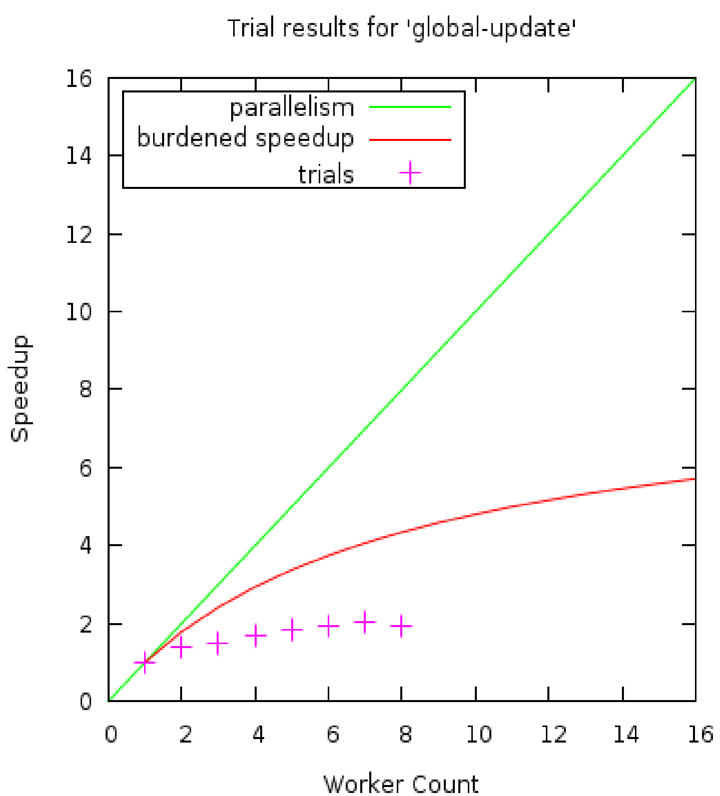
Parallelism = 36.34

Burdened Parallelism = 14.14

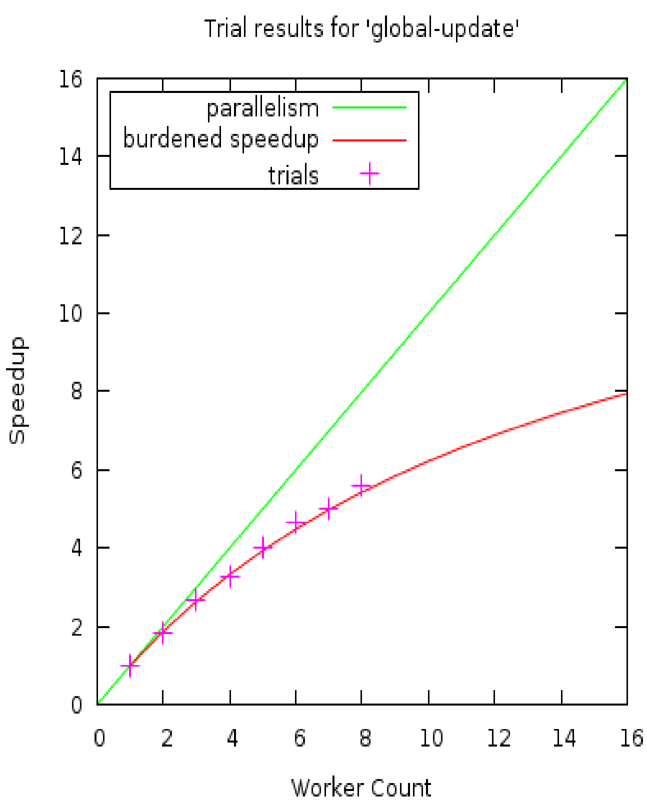
Speedup Estimate

2 procs:	1.79 - 2.00
4 procs:	2.94 - 4.00
8 procs:	4.34 - 8.00
16 procs:	5.71 - 16.00
32 procs:	6.77 - 32.00

Testing for Memory Bandwidth: extra work



Parallelism = 36.34



Parallelism = 25.74

Testing Memory Bandwidth: Running 8 Independent Copies of Serial Code

- 1 copy serial code alone: 7.848 (s)
- 8 independent copies: accounts for factor of 2 slowdown (i.e. speedup of 2 instead of 4)

copy #	update running time (s)
1	16.351
2	13.556
3	18.031
4	18.963
5	18.166
6	17.479
7	17.607
8	11.704

Concurrent Global Relabeling Heuristic

- all processors have to be suspended in order to do global relabeling – instead we should run it *concurently* with push-relabel
- Anderson and Setubal '92 introduced the concept of a *global relabeling* ***wave***
- each vertex stores a wave number – the global-relabeling wave that most recently updated it
- we only push flow between vertices with same wave number; both nodes need to be locked
- no distance relabeling operation should decrease the distance label of a node; node should be locked during relabel and global-relabeling operations

Parallel Push-Relabel

- parallel discharge in approximate highest-label first order:
  - discharge-chain
  - coarsened-discharge
  - local-queues

    [keep a local list of activated nodes]
- lock-free push-relabel

Discharge-Chain

- spawn a discharge-chain: let the processor proceed discharging its newly activated node with the highest distance label – if it exists and if its distance is >= to the global highest distance of an active node

$$\text{MAIN}_{discharge-chain}()$$
$$\quad \textbf{while } ActiveNodeSet \neq \emptyset$$
$$\qquad u \leftarrow max_{d(v)}\{v | v \in ActiveNodeSet\}$$
$$\qquad \texttt{cilk\_spawn } \text{DISCHARGE-CHAIN}(u)$$

Coarsened-Discharge

- gather a batch of active nodes to discharge into an array starting from the highest-label bucket, run a *cilk_for* loop over these nodes
- number of nodes gathered, *T*, can be varied to improve performance

$\text{MAIN}_{coarsened-discharge}()$

while $ActiveNodeSet \neq \emptyset$

// Grab the top T active nodes

$a \leftarrow buckets[top\ T\ elements]$

`cilk_for` $u \in a$

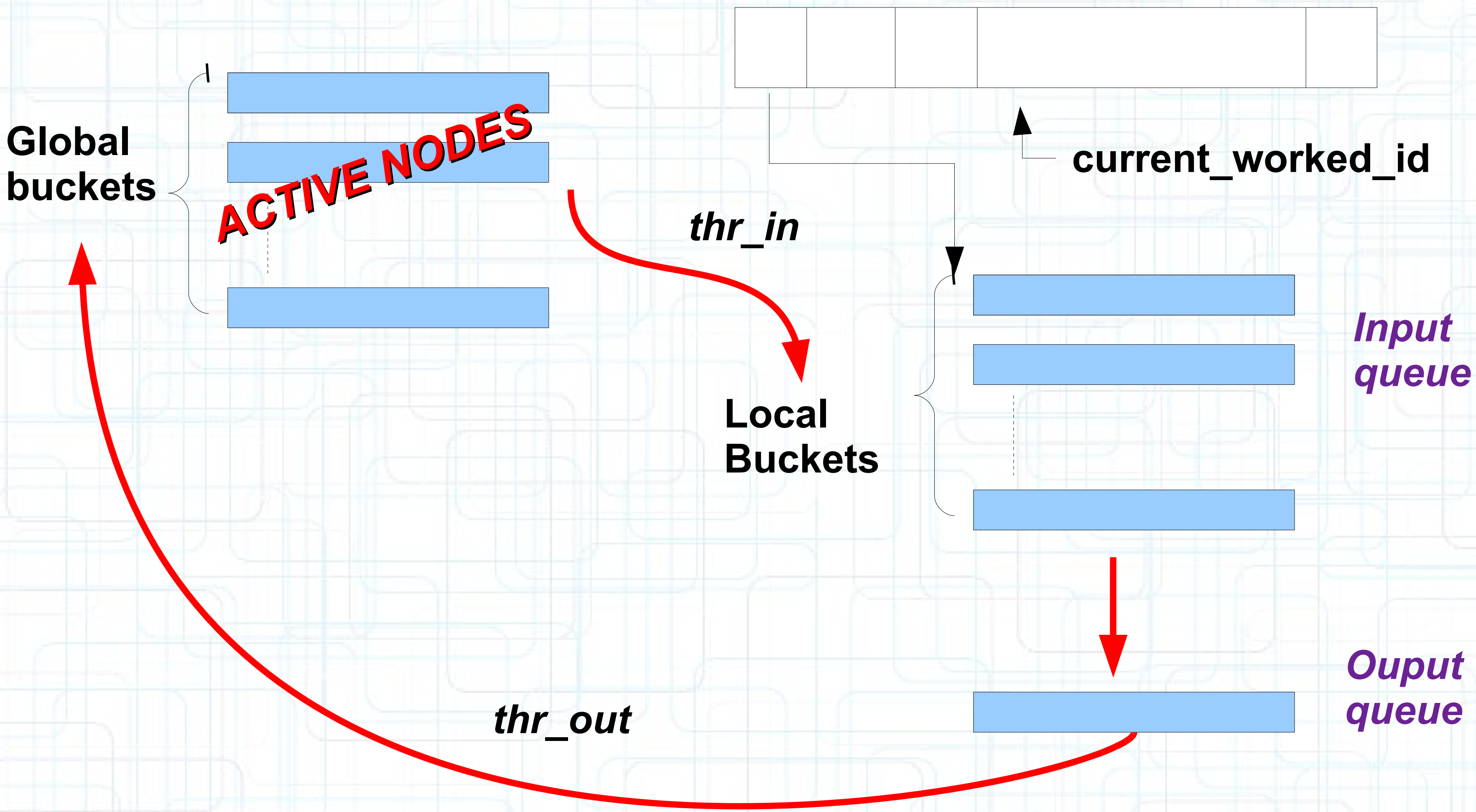
DISCHARGE(u)

In-Out Local Thread Queues (*Anderson and Setubal '92; Bader '06*)

- each thread has a local input queue of buckets and a local output queue
- threads grab active nodes to discharge from *global* buckets and place newly activated nodes into their local output queue
- when output queue is filled, the nodes in the output queue are transfered back to the global buckets
- Variables (need to be adjusted dynamically):
  - *thr_in* = how many active nodes to grab
  - *thr_out* = size of the output queue / when to sync with the global buckets

  *current implementation needs to be optimized

In-Out Local Thread Queues



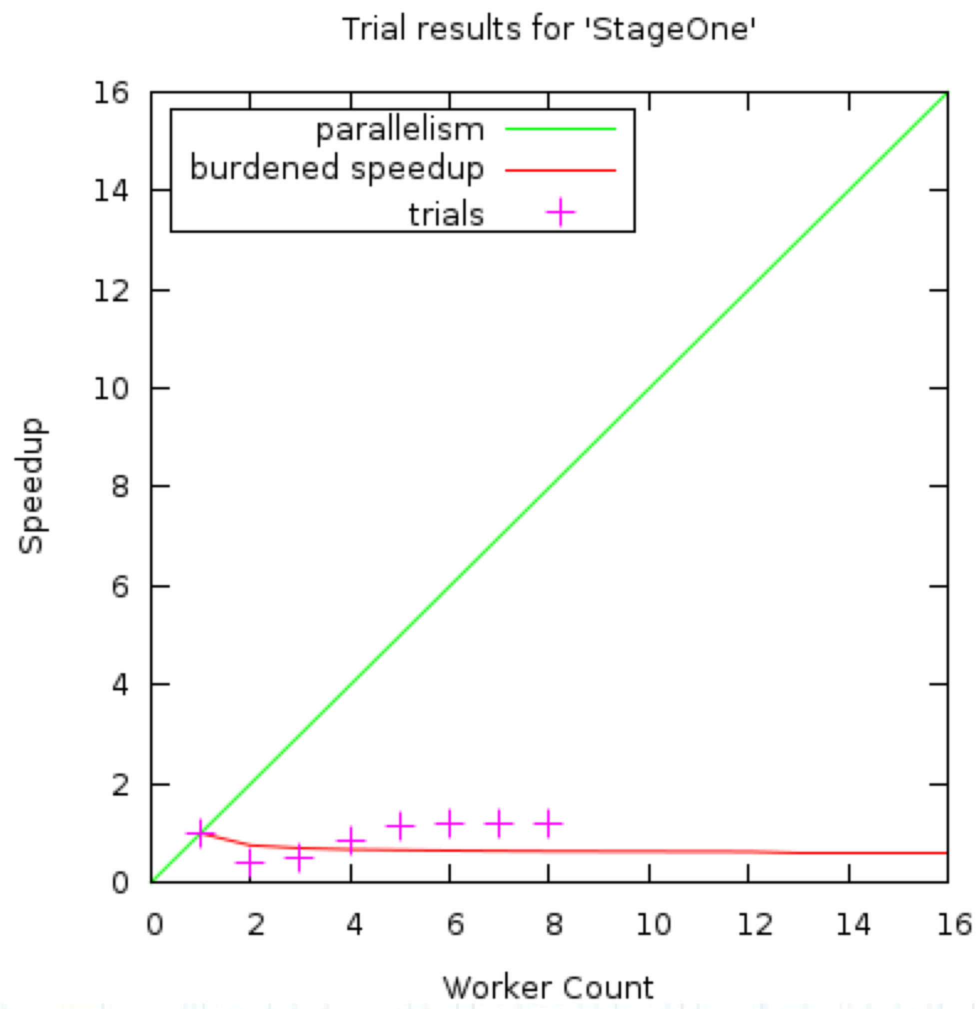
Parallel Push-Relabel Results

	algorithm	sequential	parallel	speedup
rmfl	discharge-chain	126.63	108.98	1.16
	discharge-chain-concurrent	131.8	54.07	**2.44**
	coarsened-discharge	85.83	116.79	1.16
	local-queues	176.85	166.31	1.064
rmfw	discharge-chain	94.44	86.11	1.1
	discharge-chain-concurrent	116.35	65.57	**1.77**
	coarsened-discharge	102.24	133.3	0.77
	local-queues	186.8	202.51	0.92

Running times (in seconds) of the parallel push-relabel algorithms.

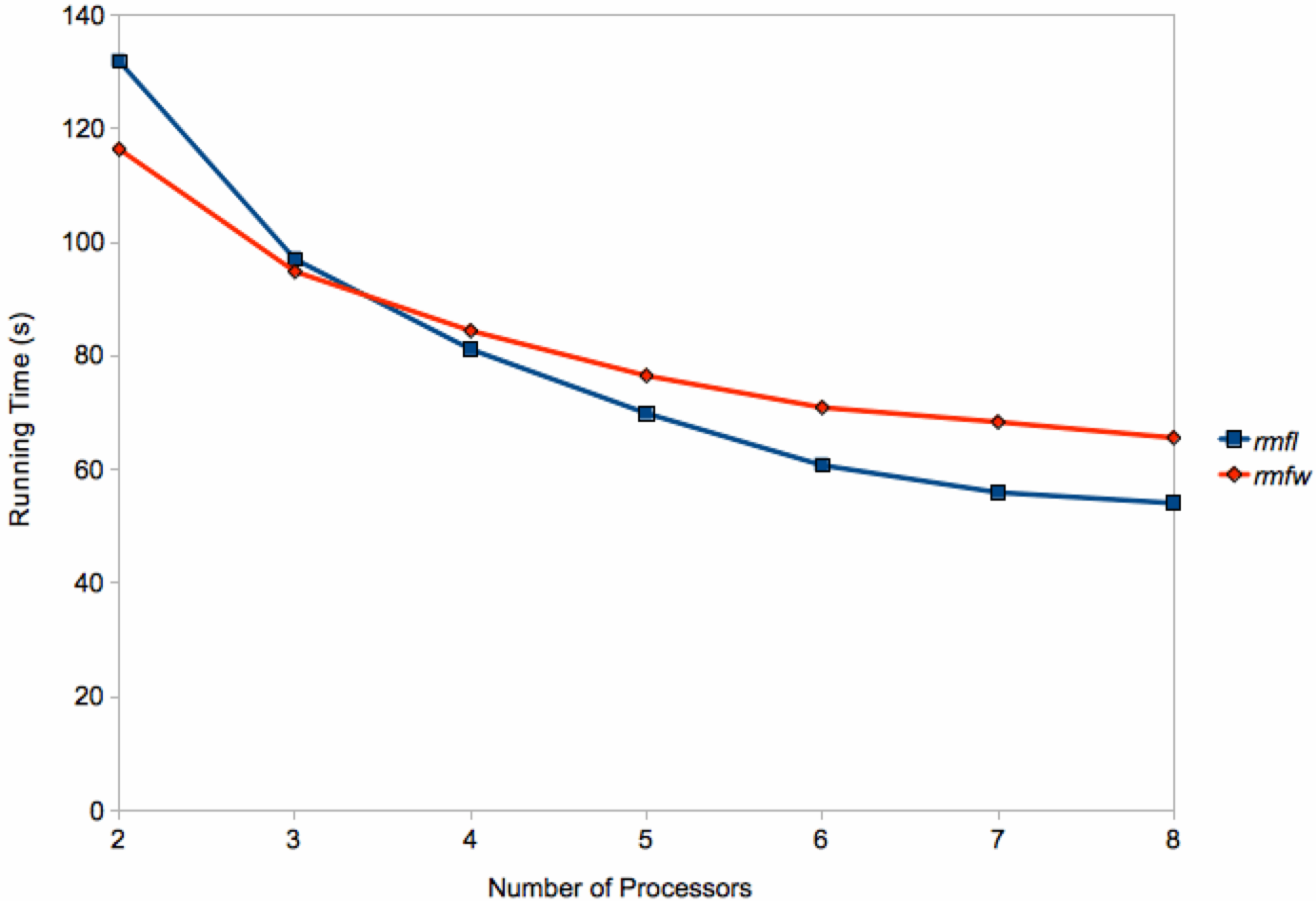
- Parallel times were obtained on 8 workers.
- *rmfl* grpah, a=50 and b = 1000, has 2,500,000 nodes and 12,297,500 edges; *rmfw* graph, a=200 and b=50, has 2,000,000 nodes and 9,920,000 edges.
- The *hipr* algorithm runs in **88.77** s on rmfl and **126.66** s on rmfw

Discharge-Chain Results



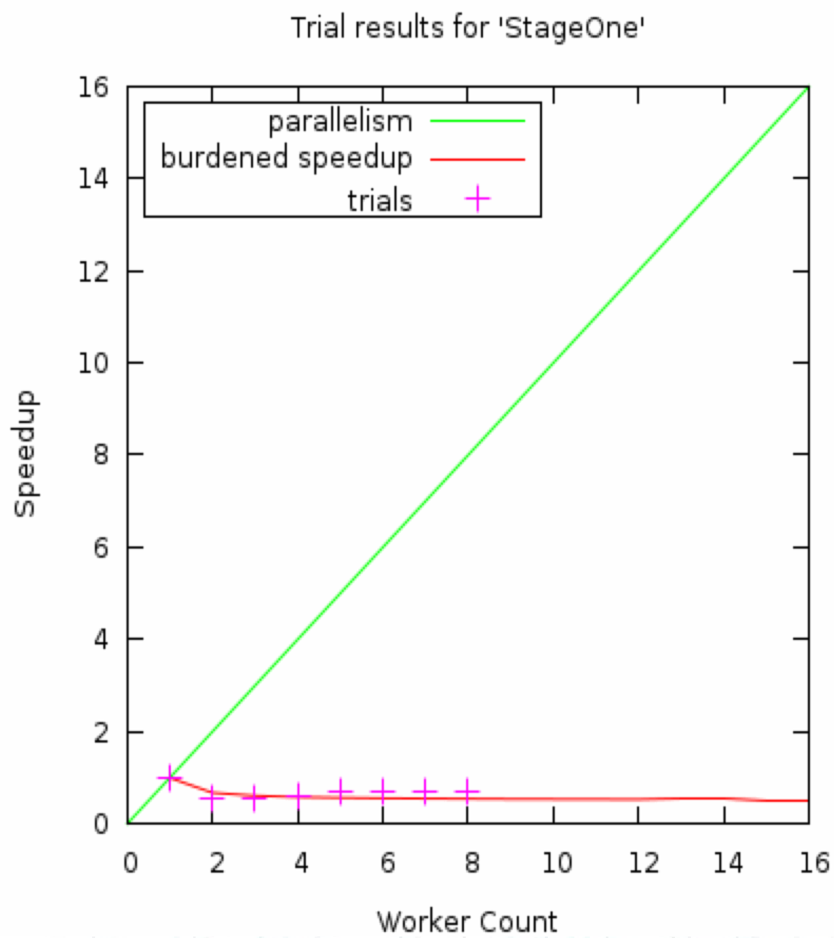
Cilkview plot: speedup for parallel push-relabel using discharge-chain on rmf(a = 50, b = 1000) **without** concurrent global-relabeling

Best: Discharge-Chain with Concurrent Global-Relabeling



Parallel push-relabel using discharge-chain with ***concurrent*** global-relabeling: speedup of ~2 on *rmfl* graphs

Coarsened-Discharge Results



Cilkview plot: speedup for parallel push-relabel using coarsened-discharge on rmf(a = 50, b = 1000) **without** concurrent global-relabeling

Lock-Free Push-Relabel (Hong'08)

- Push only to the **'lowest'** neighbor
- Lift yourself if no lower neighbor
- Done completely in parallel (per node!)
- Except .... Termination is a problem
  - Must figure out when no node has any excess
  - This now requires a barrier (aka a Lock!)
- Oh, and tons of Compare-And-Swap ops.

Lock-Free: Exactly *How* Bad?

- Original Push-Relabel : $O(N^2 E)$
- "Lock Free" (without termination): $O(N^2 E)$
- Highest Active Nodes First (hi_pr): $O(N^2 E^{1/2})$
- Tarjan Dynamic Trees: $O(N^2 \log(N^2 / E))$

- $E^{1/2}$ Slower, but potentially N Parallelism

Lock-free: Push-Uplift

PUSH$_{lockfree}(u,v)$

// Applicability: $excess(u) > 0$
$\delta = min(excess(u), c_f(u,v))$
FETCH-AND-SUBTRACT$_{atomic}(f(u,v) - \delta)$
FETCH-AND-ADD$_{atomic}(f(v,u) + \delta)$
$e_u =$ SUBTRACT-AND-FETCH$_{atomic}(excess(u) - \delta)$
$e_{v_{old}} =$ FETCH-AND-ADD$_{atomic}(excess(v) + \delta)$
if $e_{v_{old}} = 0 \;\&\; \delta > 0 \;\&\; u \notin \{source, sink\}$
  // v gained positive exces sand became active
  FETCH-AND-ADD$_{atomic}(active\text{-}node\text{-}count + 1)$
  LOCAL-ADD-TO-STRATA-OUTSET(v)

if $e_u = 0 \;\&\; \delta > 0$
  // u just became inactive
  FETCH-AND-SUBTRACT$_{atomic}(active\text{-}node\text{-}count - 1)$

UPLIFT$_{lockfree}(u)$

// Applicability: $excess(u) > 0 \;\&\; \forall(u,v)|c_f(u,v) > 0,$
// $d(u) \geq d(v)$
// First, find min distance of admissible arc neighbors
$h \leftarrow min\{d(v)|(u,v) \in G \;\&\; c_f(u,v) > 0\}$
if $\{v|(u,v) \in G \;\&\; c_f(u,v) > 0\} = \emptyset$
  return $false$
else
  $d(u) \leftarrow h + 1$
  return $true$

Lock-Free: Order Heuristic – STRATA Data Structure

```
STRATA:
    // number of layers in array
    long num-layers
    // the distance of lowest layer no including bottom
    long lowest-layer
    // max distance of any node in outset
    long max-distance
    // min distance of any node in outset
    long min-distance
    std::vector<node*> top
    std::vector<node*>* layers
    std::vector<node*> bottom
    // num elements in layers not including outset
    long num-elements
    // nodes which are locally in this strata
    // but are not in layers yet
    std::vector<node*> outset
    // num layers which will be operated in parallel
    // not including top
    long num-active-layers
    // bookeeping variable with the last
    // noderange mapping to this strata
    long node-range-last
```

Lock-Free Results