Employing Transactional Memory and Helper Threads to Speedup Dijkstra's Algorithm

Nikos Anastopoulos, Konstantinos Nikas, Georgios Goumas and Nectarios Koziris

{anastop,knikas,goumas,nkoziris}@cslab.ece.ntua.gr

Computing Systems Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens

Motivation

TM community needs real-world applications

- Graph algorithms are described as good candidates for TM, due to irregular accesses of underlying data structures

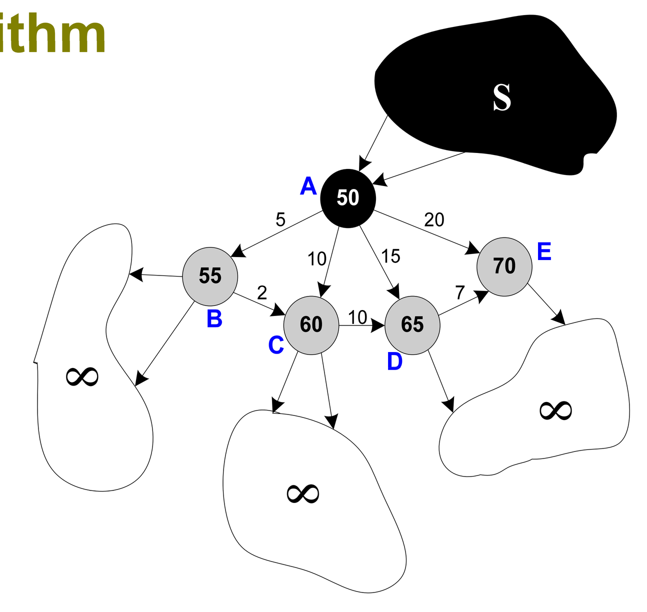
Dijkstra's algorithm

- fundamental and widely used algorithm for the single-source shortest paths (SSSP) problem
- inherently serial, thus challenging to parallelize
- previous attempts resulted in major changes in algorithm's semantics or structure (e.g. Δ-stepping, Boost library implementation)

The Basics of Dijkstra's Algorithm

Serial algorithm

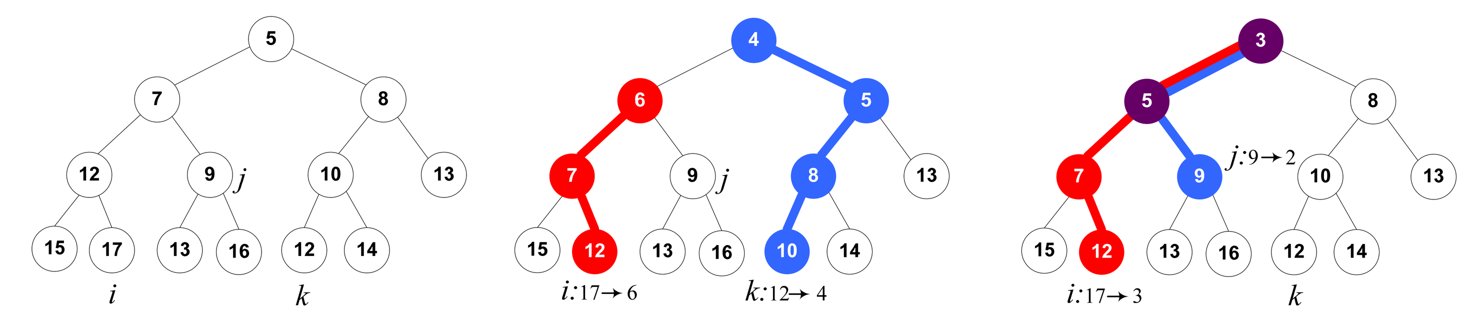
```
Input    : G=(V,E), w: E→R+,
           source vertex s, min Q
Output   : shortest distance array d,
           predecessor array π
foreach v ∈ V do
    d[v] ← INF;
    π[v] ← NIL;
    Insert(Q,v);
end
d[s] ← 0;

while Q ≠ empty do
    u ← ExtractMin(Q);
    foreach v adjacent to u do
        sum ← d[u] + w(u,v);
        if d[v] > sum then
            DecreaseKey(Q,v,sum);
            d[v] ← sum;
            π[v] ← u;
        end
    end
end
```



For each vertex v the algorithm maintains a *shortest path estimate* d(v), which gradually converges to the actual distance δ(v), through *edge relaxations*

Three partitions of vertices

- **Settled**: d(v) = δ(v)
- Queued: d(v) > δ(v) and d(v) ≠ ∞
- Unreached: d(v) = ∞



Min-priority queue implemented as a binary min-heap

- maintains all but the settled ("optimal") vertices
- min-heap property: *for each i, d(parent(i)) ≤ d(i)*
- amortizes the cost of multiple ExtractMin's and DecreaseKey's

Straightforward Parallelization

Fine-Grain Multi-Threaded

```
/*Initialization phase same to serial code */
while Q ≠ empty do
    Barrier
    if tid = 0 then
        u ← ExtractMin(Q);
    Barrier
    for v adjacent to u in parallel do
        sum ← d[u] + w(u,v);
        if d[v] > sum then
            Begin-Atomic
            DecreaseKey(Q,v,sum);
            End-Atomic
            d[v] ← sum;
            π[v] ← u;
        end
end
```

Issues:

- speedup bounded by average out-degree
- concurrent heap updates to due DecreaseKey's
- barrier synchronization overhead (more than 70% of total execution time)

Evaluation

- conventional synchronization mechanisms yield major slowdowns
- TM gives better performance, highlights optimistic parallelism, but still suffers from barriers overhead

Helper-Threading Scheme

Motivation

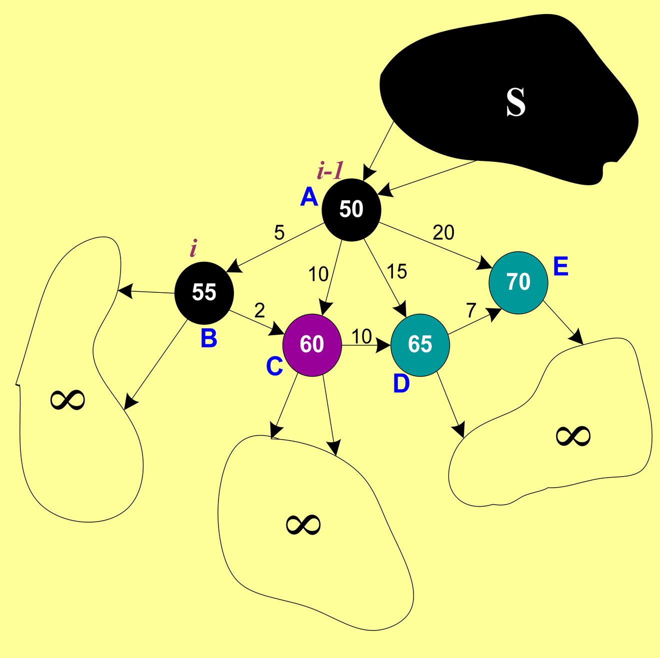
- expose more parallelism to each thread
- eliminate costly barrier synchronization

Rationale

- in serial, updates are performed only from *definitely* optimal vertices
- allow updates from *possibly* optimal vertices
  - main thread operates as in serial
  - helper threads are assigned the next minimum vertices (x_k) and perform updates from them

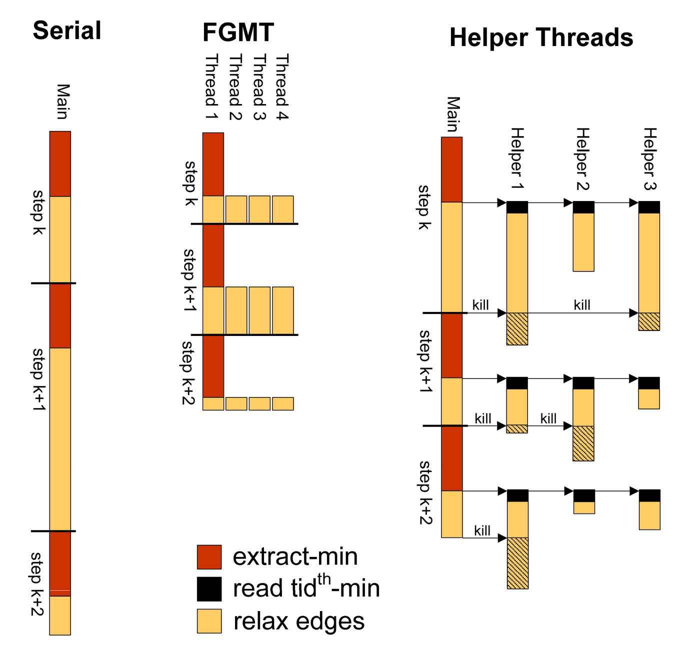
Speculation on the status of x_k

- if already optimal, main thread will be offloaded
- if not optimal, any suboptimal relaxations will be corrected eventually by main thread



Example of HT scheme. In step i-1, node A was extracted and its neighbours were relaxed to the values shown. In step i, the main thread extracts B while the helpers are assigned the next 3 nodes in the queue (C,D,E). For D and E, the helpers perform correct relaxations, since in step i-1 they obtained their final distances. As a result, the main thread will be offloaded from these relaxations. On the contrary, the helper that took over C will perform suboptimal relaxations, since at the end of step i, C's distance will be updated to 57 by the main thread. However, in step i+1 the main thread will extract C and relax again its out-edges, using now the correct distance.

Execution Patterns



Implementation of HT-Scheme

Main thread

```
while Q ≠ empty do
    u ← ExtractMin(Q);
    done ← 0;
    foreach v adjacent to u do
        sum ← d[u] + w(u,v);
        Begin-Xact
        if d[v] > sum then
            DecreaseKey(Q,v,sum);
            d[v] ← sum;
            π[v] ← u;
        End-Xact
    end
    Begin-Xact
    done ← 1;
    End-Xact
end
```

Helper thread

```
while Q ≠ empty do
    while done = 1 do ;
    x ← ReadMin(Q,tid);
    stop ← 0;
    foreach y adjacent to x and while stop = 0 do
        Begin-Xact
        if done = 0 then
            sum ← d[x] + w(x,y);
            if d[y] > sum then
                DecreaseKey(Q,y,sum);
                d[y] ← sum;
                π[y] ← x;
        else
            stop ← 1;
        End-Xact
    end
end
```

Why with TM?

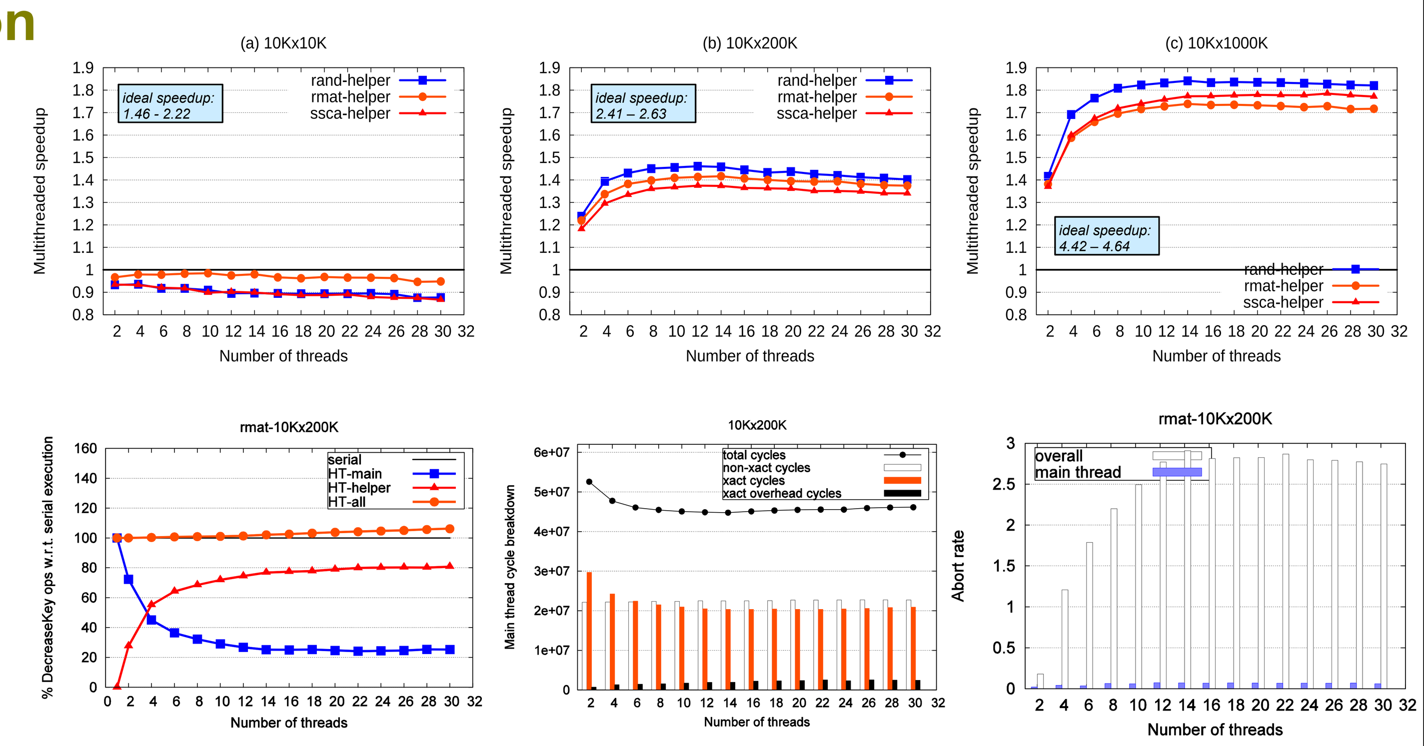
- composable: all dependent atomic sub-operations (check for relaxation, heap updates, d,π updates) composed into a large atomic operation, without limiting concurrency
- optimistic
- easily programmable
- offers a way for helper threads interruption

Experimental Evaluation

- Simics 3.0.31, GEMS 2.1
- LogTM-SE (EE, *HYBRID* resolution policy)
- 3 different graph families, 6 graph densities

Results

- speedups in 15 out of 18 cases (not all shown), up to 1.84
- main thread not obstructed by helpers (<1% abort rate)



Future work

TM for optimistic *parallelization*

- HT+TM as a programming model for other graph problems (MSTs, maximum flow, SSSP) and other similar ("greedy") applications
- adjustments of existing TM systems for explicitly supporting speculative parallelization

(or, what if we didn't rely on Dijkstra's algorithm semantics to correct things?)

Publications

- *"Employing Transactional Memory and Helper Threads to Speedup Dijkstra's Algorithm"*. K. Nikas, N. Anastopoulos, G. Goumas, N. Koziris. In Proc. of the International Conference on Parallel Processing (ICPP 2009), Vienna, Austria, September 22-25, 2009. (to appear).
- *"Early Experiences on Accelerating Dijkstra's Algorithm Using Transactional Memory"*. N. Anastopoulos, K. Nikas, G. Goumas, N. Koziris. In Proc. of the 3rd Workshop on Multithreaded Architectures and Applications (MTAAP 2009), Rome, Italy.