Multithreaded distributed BFS with regularized communication pattern on the cluster with Angara interconnect

T-PLATFORMS

March 3, 2016

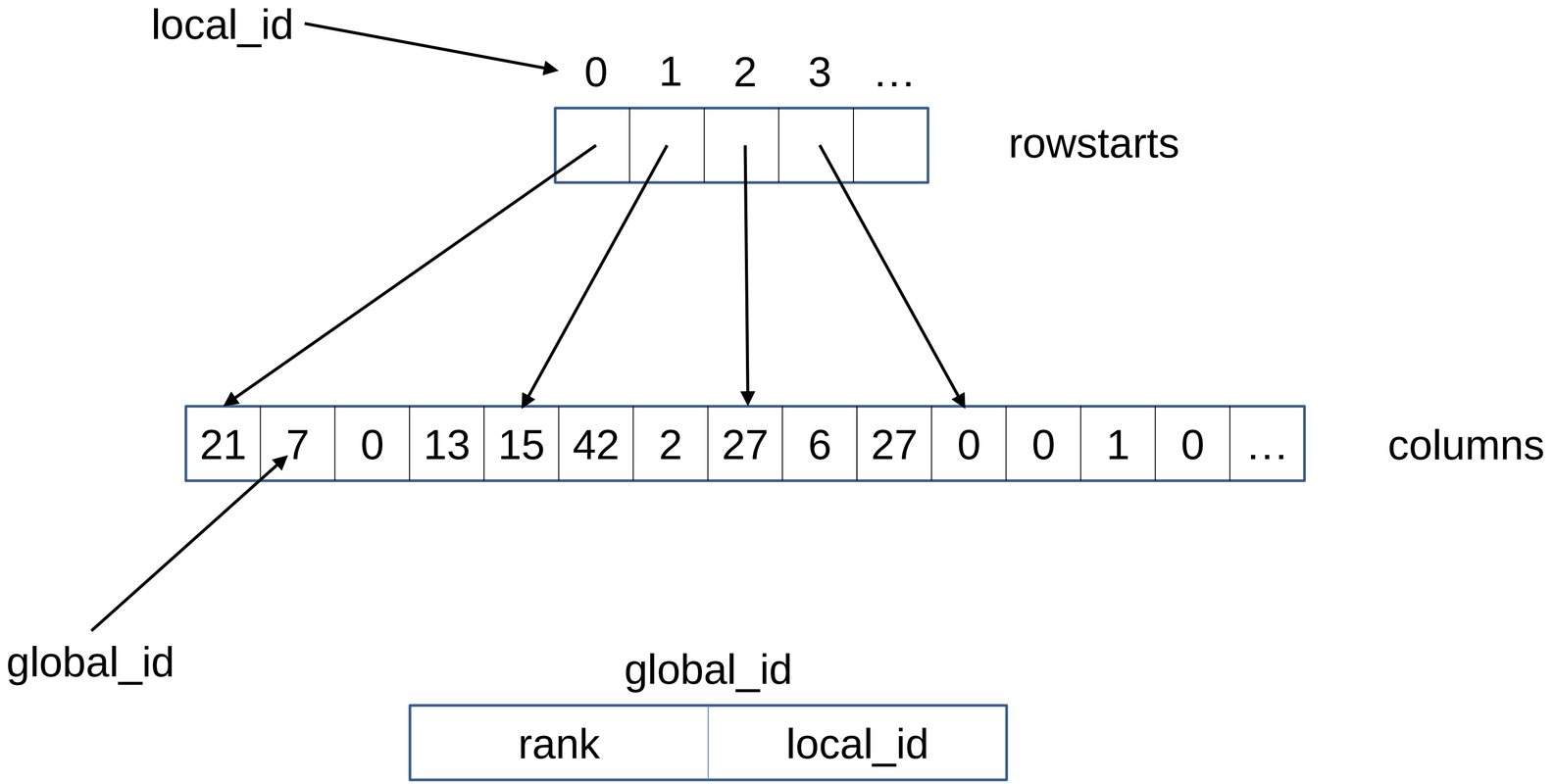
Artem Osipov
Alexander Daryin

BFS Revisited

- **Breadth-First Search (BFS)** on distributed memory systems
- Part of Graph500 benchmark

- Performance is limited by:
  - memory access latency
  - network latency
  - network bandwidth

Compressed Sparse-Row Graph (CSR)



Reference Algorithm

Data: CSR (rowstarts R, column C), current queue Q

```
function ProcessQueue(R, C, Q)
  for v in Q do
    for e in {R[v]..R[v + 1]} do
      send v, local(C[e]) to owner(C[e]))
```

Reference Algorithm

Data: CSR (rowstarts R, column C), current queue Q

```
function ProcessQueue(R, C, Q)
  for v in Q do
    for p in {R[v]..R[v + 1]} do
      send v, local(C[p]) to owner(C[p])
```

Small message size
low bandwidth

Irregular communication pattern
connection overhead

Message coalescing

Separate send buffer for each peer process
large memory overhead
multiple threads - ?

Goals

- Optimize memory access
- Regularize communication pattern
- Reduce memory consumption
- Allow for multithreaded design

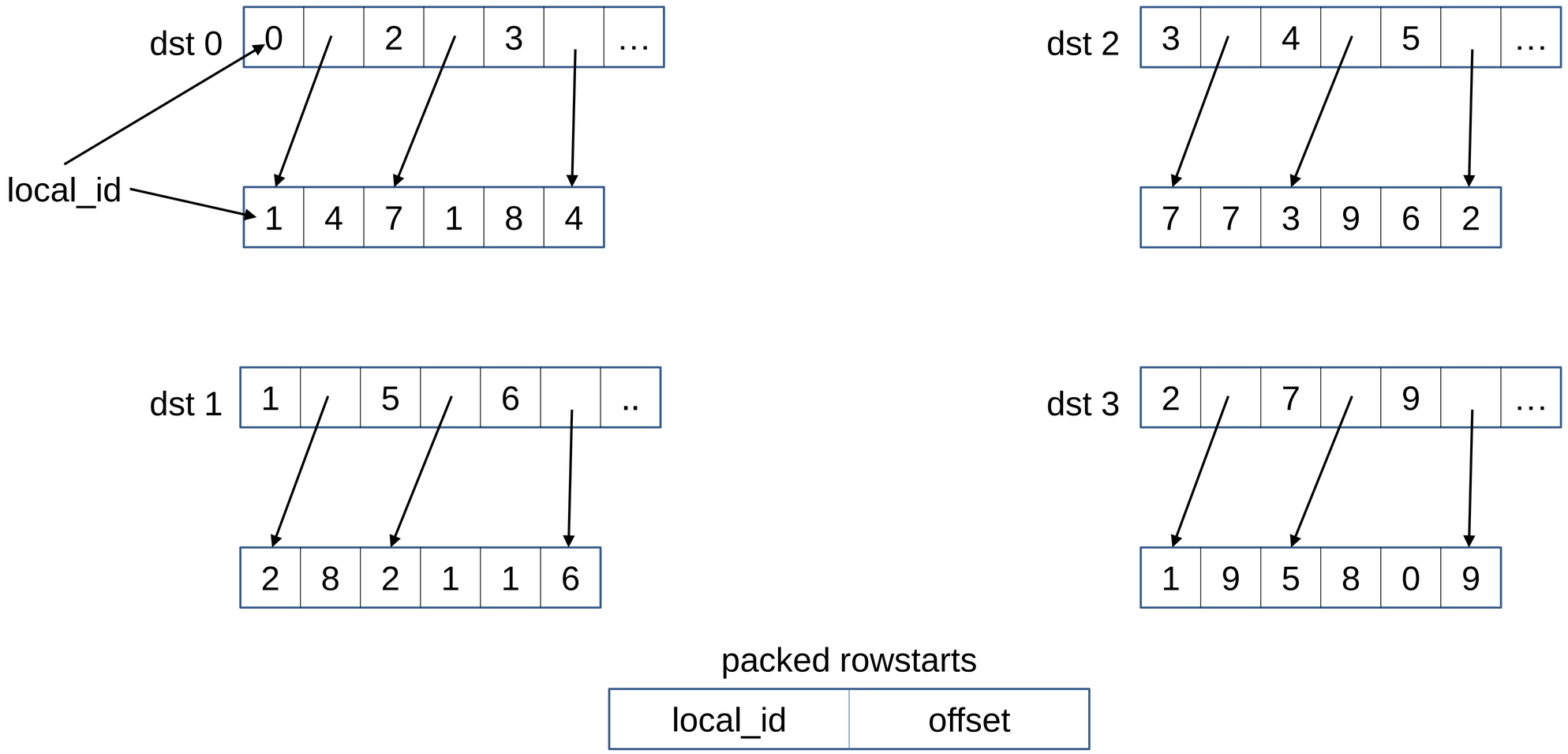
- **Possible solution:**
  - partition graph data for each peer node
  - process partitions independently (and possibly concurrently)

Regularized Communications Pattern

- Partition graph data for each peer node
- Process partitions independently (and concurrently)

- **Problem:**
  - Unacceptable memory overhead when using standard CSR representation - |rowstarts| = |V|
- **Solution:**
  - Pack rowstarts

Packed Partitioned CSR



Packed CSR Algorithm

Data: Packed CSR (vertex indices V, row offsets D, column C), current queue Q (bit mask), number of processes NP

```
function ProcessQueue(R, C, Q)
  for p in {0..NP-1} do
    for i in {0..|V[p]|-1} do
      if V[p][i] in Q then
        for e in {D[p][i]..D[p][i + 1] do
          send V[p][i], C[e]) to p
```

Direction Optimization

- **Standard algorithm:** when sending (u, v) edge for already visited u we're hoping that v is unvisited
- At some point it becomes easier to hit visited v than unvisited

- **Backward stepping algorithm:** send (u, v) edge for unvisited u and hope that v is visited
- Reduce communications – use edge probing
- To increase the effectiveness of probing sort columns by degree

Backwards Stepping

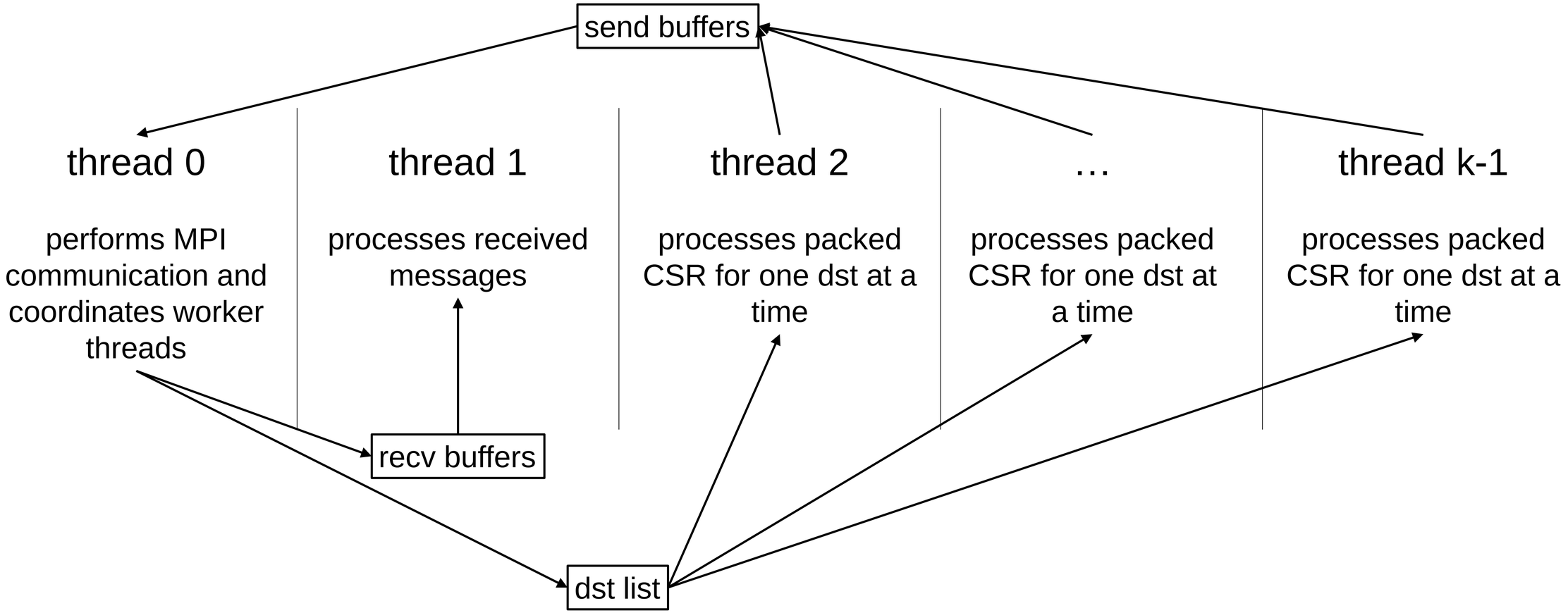
Data: Packed CSR (vertex indices V, row offsets D, column C), bit mask of visited vertices M, number of processes NP

```
function ProcessUnvisited()
  // probe first edges
  for p in {0..NP-1} do
    for i in {0..|V[p]-1|} do
      if V[p][i] in M then
        send V[p][i], C[D[p][i]] to peer
    flush send buffer
  wait for all acks
  // probe other edges
  for p in {0..NP-1} do
    for i in {0..|V[p]|-1} do
      if V[p][i] in M then
        for e in {D[p][i] + 1..D[p][i + 1]} do
          send V[peer][i], C[e] to peer
    flush send buffer
```

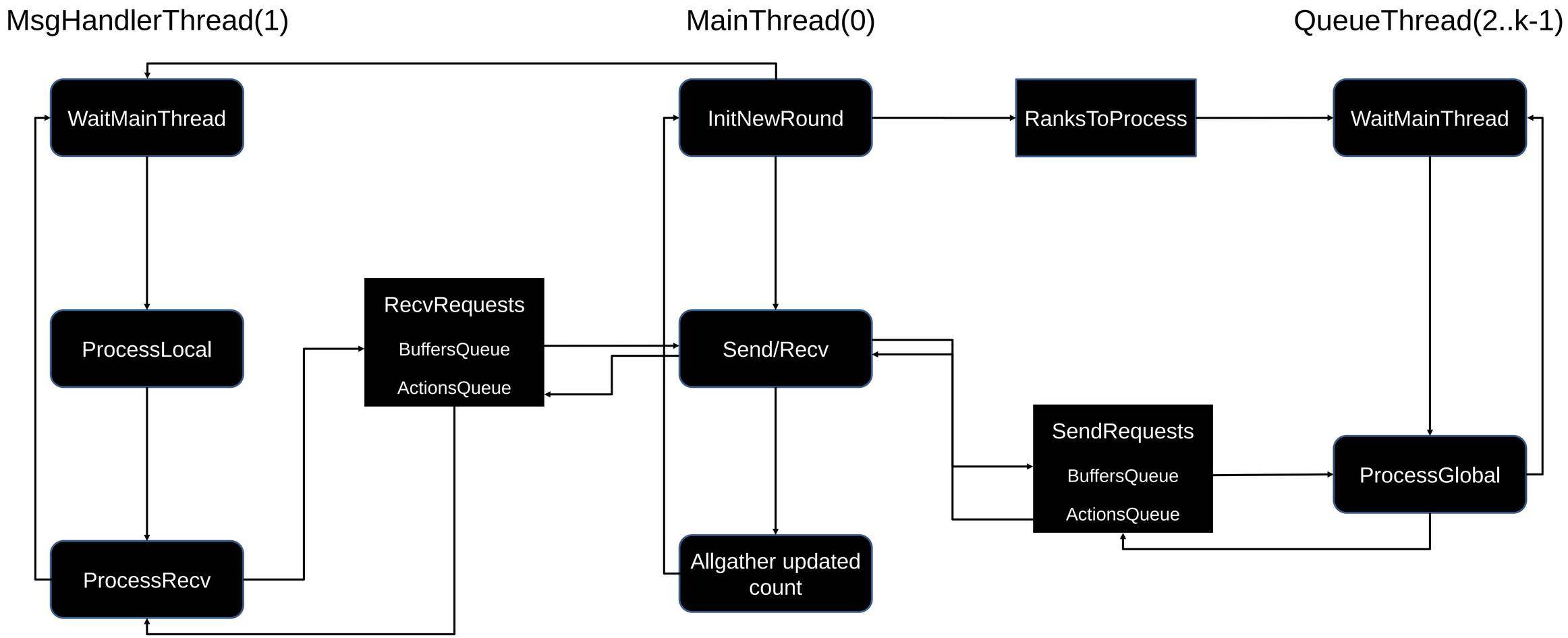
Multithreading

- **Basic concepts:**
  - All MPI communications go through the main thread
  - All received data is handled by dedicated thread
  - Other threads prepare data to send
  - All communications between threads go through concurrent queues (Intel Thread Building Blocks package was used)

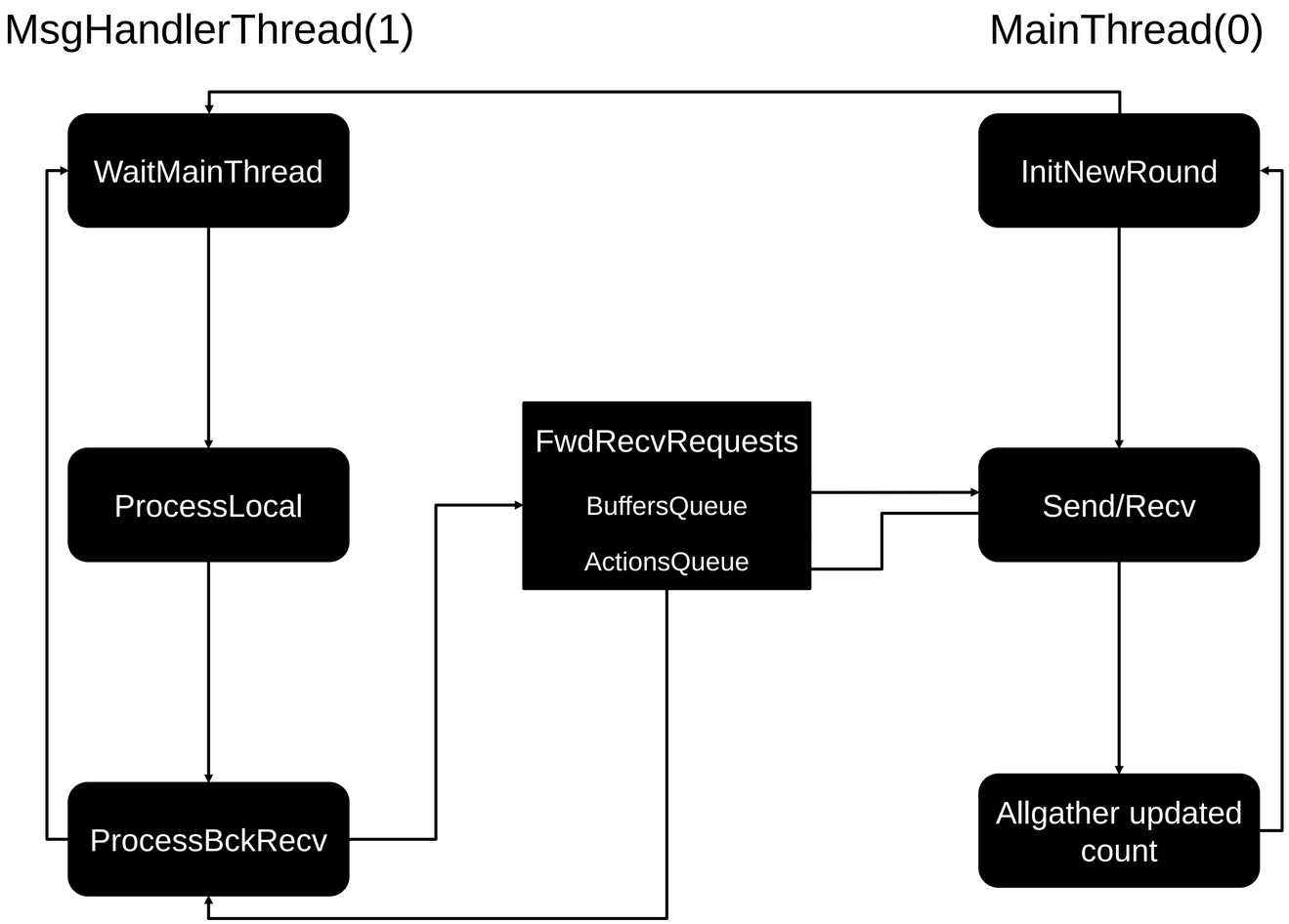
Multithreading



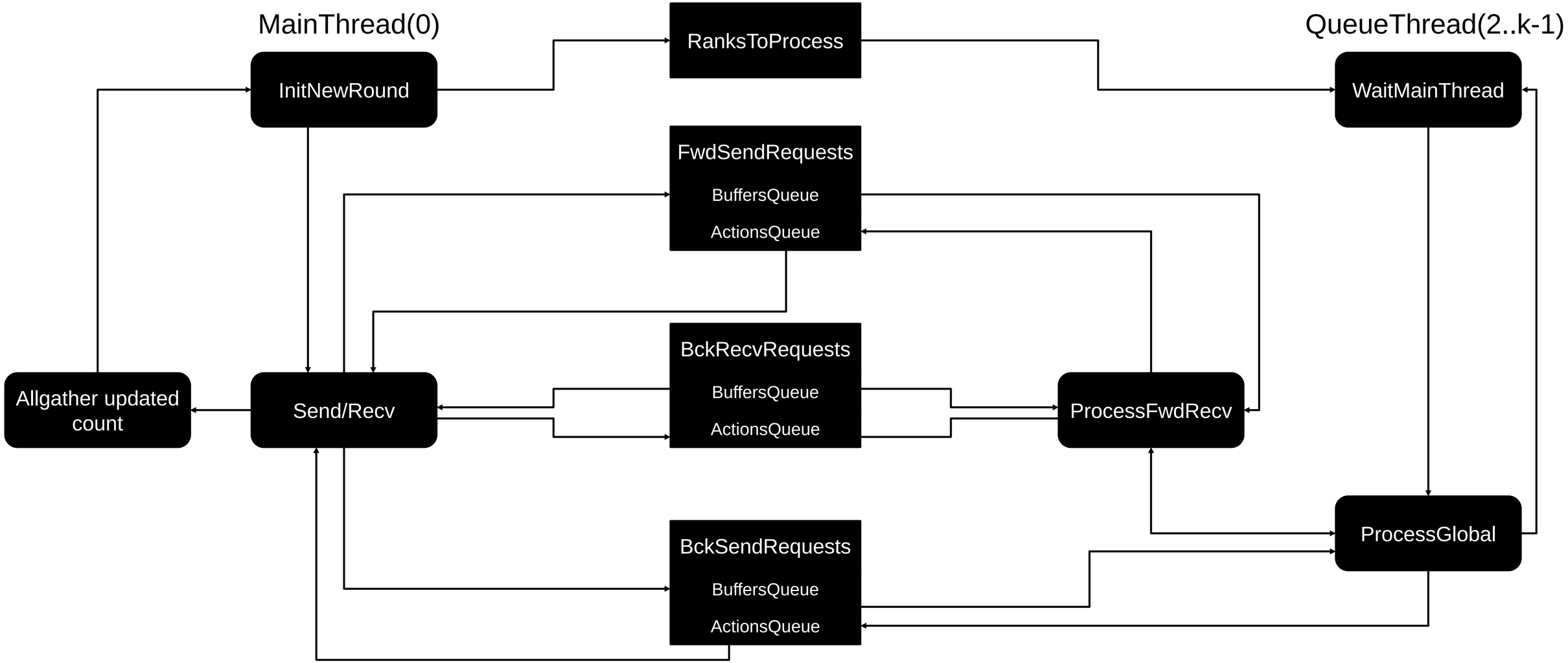
Forward stepping



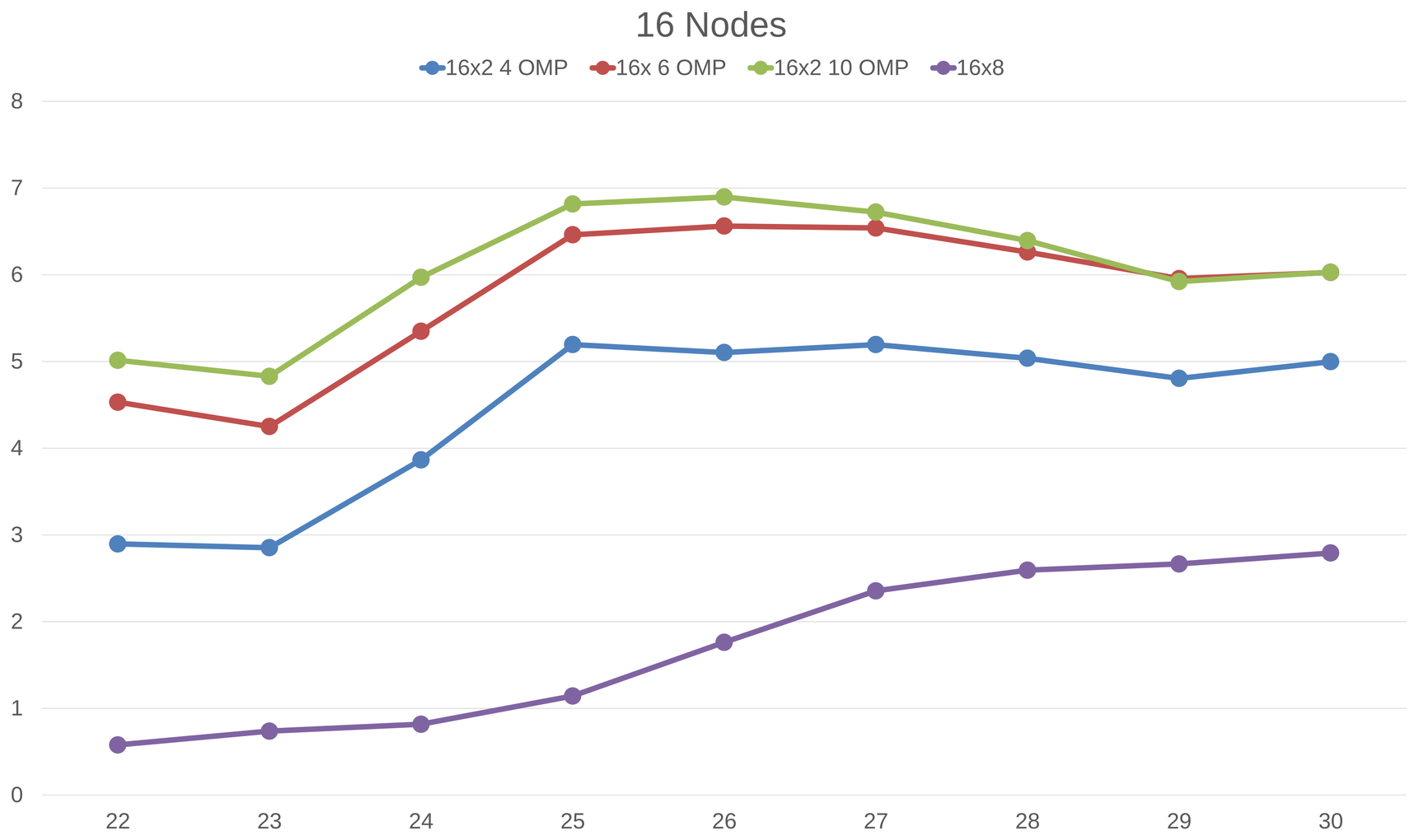
Backward stepping



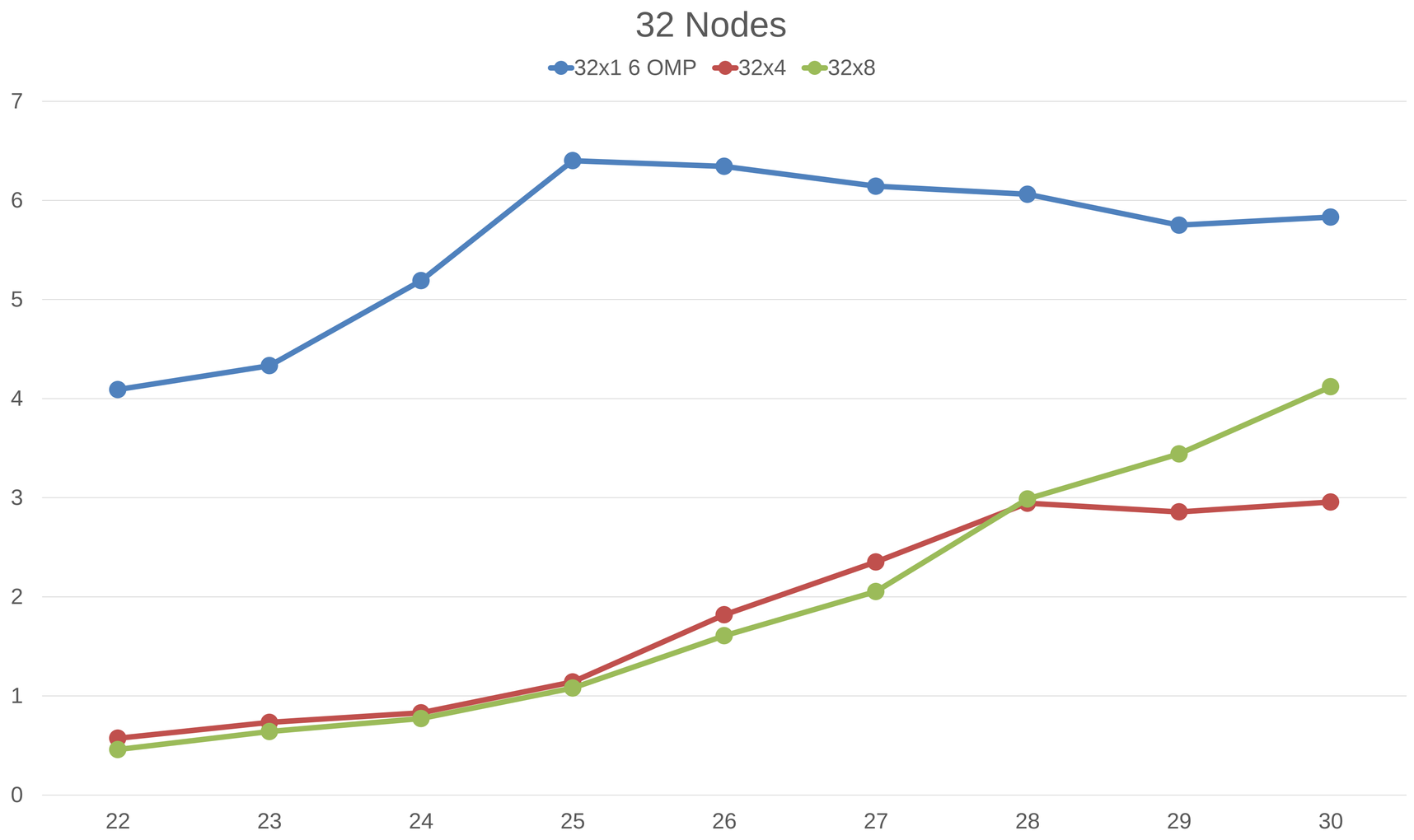
Backward stepping



Performance: “Angara K1”



Performance: “Angara K1”



Future optimizations

- Graph redistribution (maximal clustering per node)
- Multithreaded heavy

THANK YOU!

www.t-platforms.com