# Scalable and Fault Tolerant Orthogonalization Based on Randomized Aggregation

**W. N. Gansterer**, G. Niederbrucker, H. Strakova
and S. Schulze Grotthoff

Research Lab Computational Technologies and Applications
Research Group Theory and Applications of Algorithms
University of Vienna, Austria

# Objective

## Distributed algorithms for matrix computations

- Decentralized
- Nodes operate (mostly) with local information
- Nodes do not need to be synchronized
- Automatically adapt to arbitrary topologies (incl. changes during runtime)

$\Rightarrow$ Less synchronization, less global information than classical parallel algorithms

$\Rightarrow$ Dynamically changing communication schedules

## Which potential do they have in terms of

- Scalability with system size ?
- Resilience ?

# Building Block

## Distributed Data Aggreggation Algorithms (DDAAs)

- Reduction operations (summation, averaging, etc.)
- Based on (randomized) gossiping protocols:

---

```
loop (in node k)
    send data to randomly chosen neighbor
    if received data ⇒ update local data
end loop
```

---

- Well established in distributed systems, sensor networks, telecommunications, etc.

# Basic Approach

Three levels:

- DDAAs
- Distributed BLAS operations
- Matrix computations

Case study in this talk: orthogonalization / QR decomposition

# Outline

Push-sum [Kempe et al. 2003]

- Send half of the local value $x_i$ to a randomly chosen node
- Update a weight $w_i$ such that $x_i / w_i$ is the local estimate
- Guaranteed to converge linearly to sum or average

LiMoSense [Eyal et al. 2011]

- Push-sum + history

Flow updating [Jesus et al. 2009]

- Maintain a *flow variable* for each communication link
- Local value is added to the flow variable and then the flow variable is sent to randomly chosen node
- Receiver updates its own flow variable with the negated received flow

$\Rightarrow$ Without failures, sum of flows is zero (*flow conservation*, cf. network flow algorithms)

- Recovering from a failure corresponds to (re)establishing flow conservation
- Local estimate = subtract the sum of flows maintained from initial value, then average with all local estimates of neighbors
- Convergence (speed) not formally analyzed (appears slow)

# Failure Types

(F1) *Reported temporary* unavailability of nodes/links

(F2) *Unreported* loss or corruption of a message

(F3) *Reported permanent* node/link failures

(F4) *Unreported* corruption of data (e. g., bit flip)

(F5) *Unreported permanent* node failures

# Resilience Properties of DDAAs

| | Push-sum | LiMoSense | Flow updating |
|---|---|---|---|
| (F1) | $\surd$ | $\surd$ | $\surd$ |
| (F2) | — | $\surd$ | $\surd$ |
| (F3) | — | $\surd$ | $\surd$ |
| (F4) | — | — | $\surd$ |
| (F5) | — | — | — |

Note:
Flow-based approach can also recover
from purely local failures of a node (F4)

$\Rightarrow$ Idea:

- Integrate the flow concept into push-sum !

Benefits:

- Convergence properties of push-sum
- Improved resilience due to flow concept

# The Push-Flow Algorithm

initialize: $v_i \leftarrow (x_i, 1)$, $f_{i,j} \leftarrow (0, 0)$

**for all** received pairs $f_{j,i}$ **do**

    $f_{i,j} \leftarrow -f_{j,i}$

**end for**

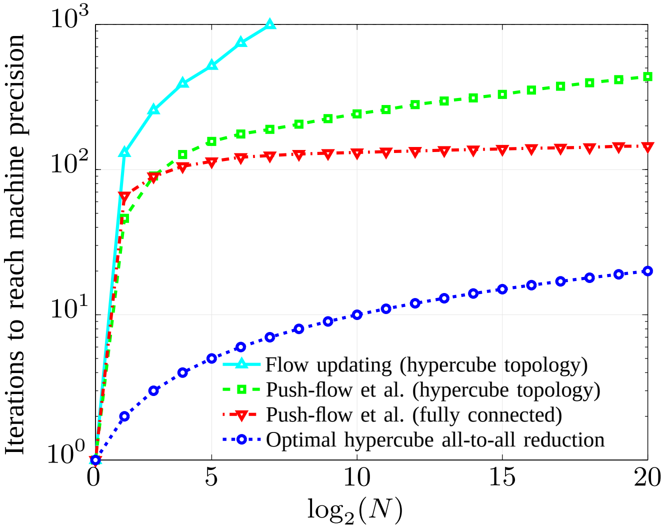
choose a random neighbor $k \in \mathscr{G}_i$

update the flow to node $k$: $f_{i,k} \leftarrow f_{i,k} + \left(v_i - \sum_{j \in \mathscr{G}_i} f_{i,j}\right)/2$

send $f_{i,k}$ to node $k$

$\mathscr{G}_i$ denotes node $i$'s neighborhood

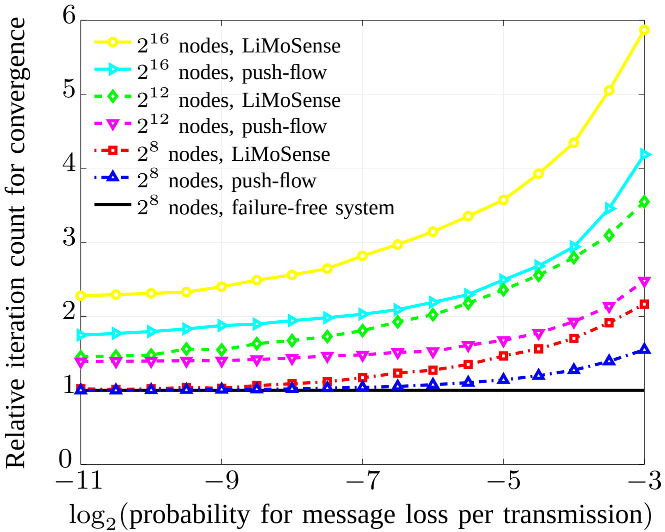
# Scaling Behavior




*N* nodes

No failures

# Scaling Behavior




$2^8 - 2^{16}$ nodes
Hypercube
Varying failure rate

# Outline

**Input:** $A \in \mathbb{R}^{n \times m}$ (*for simplified illustration n=N*)

**Output:** $Q \in \mathbb{R}^{n \times m}$, $R \in \mathbb{R}^{m \times m}$

**for** $i = 1$ **to** $m$ **do** (in node $k$)
    $x(k) = A(k,i)^2$
    $s = \sum_{l=1}^{n} x(l)$
    $R(i,i) = \sqrt{s}$
    $Q(k,i) = A(k,i)/R(i,i)$
    **for** $j = i+1$ **to** $m$ **do**
        $x(k) = Q(k,i)A(k,j)$
        $R(i,j) = \sum_{l=1}^{n} x(l)$
        $A(k,j) = A(k,j) - Q(k,i)R(i,j)$
    **end for**
**end for**

**Input:** $A \in \mathbb{R}^{n \times m}$ (*for simplified illustration n=N*)

**Output:** $Q \in \mathbb{R}^{n \times m}$, $R \in \mathbb{R}^{m \times m}$

**for** $i = 1$ **to** $m$ **do** (in node $k$)

  $x(k) = A(k,i)^2$

  $s_k = \textbf{\textit{DDAA}}(x)$

  $R_k(i,i) = \sqrt{s_k}$

  $Q(k,i) = A(k,i)/R_k(i,i)$

  **for** $j = i+1$ **to** $m$ **do**

    $x(k) = Q(k,i)A(k,j)$

    $R_k(i,j) = \textbf{\textit{DDAA}}(x)$

    $A(k,j) = A(k,j) - Q(k,i)R_k(i,j)$

  **end for**

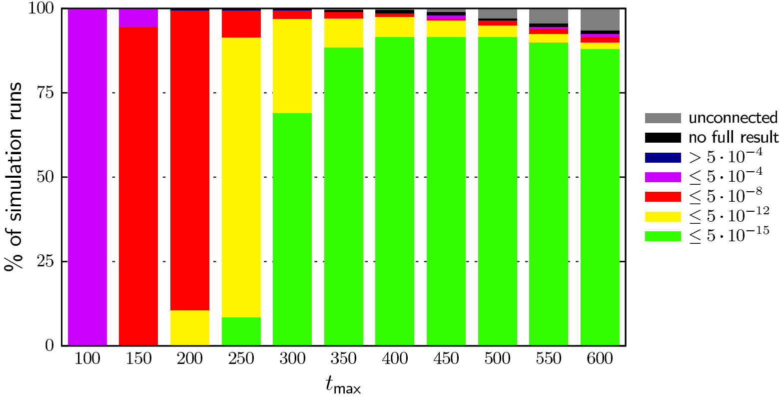
**end for**

**Input:** $A \in \mathbb{R}^{n \times m}$ (*for simplified illustration n=N*)

**Output:** $Q \in \mathbb{R}^{n \times m}$, $R \in \mathbb{R}^{m \times m}$

**for** $i = 1$ **to** $m$ **do** (in node $k$)
    . . . check for node failures, update $P_k$ and $B_k$ . . .
    $x(k) = \sum_{p \in P_k} A(p,i)^2$
    $s_k = \textbf{\textit{DDAA}}(x)$
    $R_k(i,i) = \sqrt{s_k}$
    **for each** $p \in P_k$
        $Q(p,i) = A(p,i)/R_k(i,i)$
    **for each** $b \in B_k$
        $Q(b,i) = A(b,i)/R_k(i,i)$
    **for** $j = i+1$ **to** $m$ **do**
        . . . check for node failures, update $P_k$ and $B_k$ . . .
        $x(k) = \sum_{p \in P_k} Q(p,i)A(p,j)$
        $R_k(i,j) = \textbf{\textit{DDAA}}(x)$
        **for each** $p \in P_k$
            $A(p,j) = A(p,j) - Q(p,i)R_k(i,j)$
        **for each** $b \in B_k$
            $A(b,j) = A(b,j) - Q(b,i)R_k(i,j)$
    **end for**
**end for**

# Factorization Error of rdmGS




5D hypercube, $\lambda = 12$ [s], 200 simulation runs

On average, 5.82 nodes failed per simulation run (min = 0, max = 17)

# Node Failures in Simulation

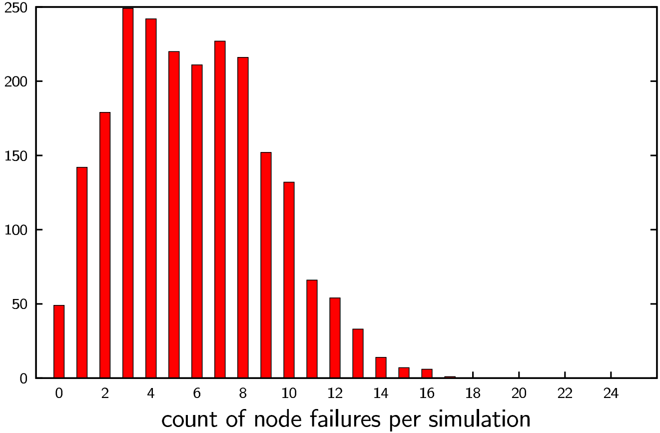



Figure: Number of node failures per simulation for $\lambda = 12[s]$ and $h = 5$ over all $t_{\max}$. Average of 5.82 node failures over all simulations

# Node Failures in Simulation

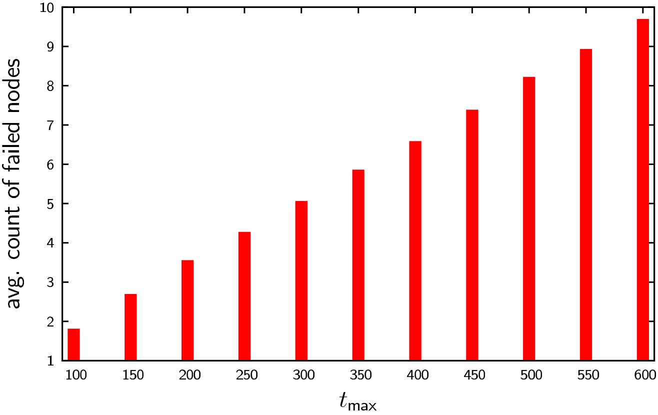
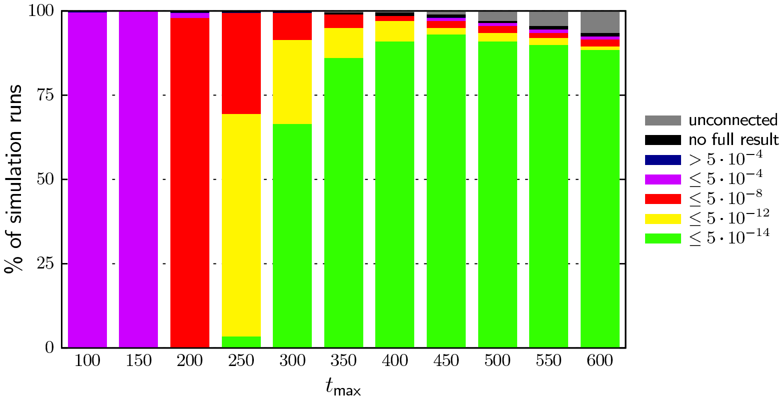



Figure: Average number of node failures per simulation for $\lambda = 12[s]$ and $h = 5$ and varying $t_{max}$. Average of 5.82 node failures over all simulations

# Orthogonality of rdmGS




5D hypercube, $\lambda = 12$ [s], 200 simulation runs

On average, 5.82 nodes failed per simulation run (min = 0, max = 17)

## Influence of the context

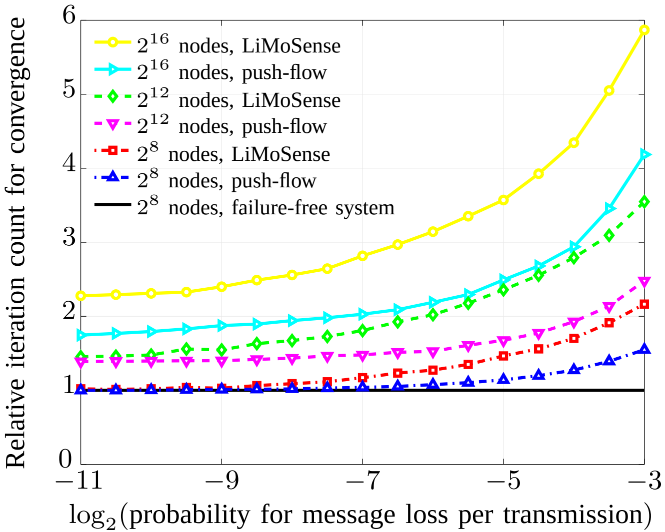
Performance (comparison) of various approaches
strongly depends on various context parameters !

- Topology
- Routing information (beyond local neighborhood)
- Properties of the failure distribution(s):
  MTBF, MTTR, etc.
- Properties of the application:
  ratio of application runtime to checkpointing interval, etc.
- ...

# Checkpointing & Restarting

- Overhead in terms of time and storage
- Often assumes restart of the application on the same number of nodes
- Coordinated vs. uncoordinated
- Stable common storage vs. distributed storage

# Overhead of DDAAs



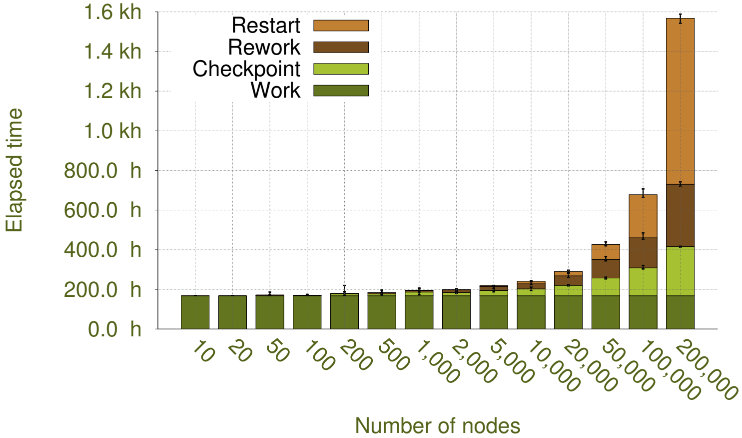

$2^8 - 2^{16}$ nodes

Hypercube

Varying failure rate

$N = 2^{16}$: $\leq 2$ for low failure rates

# Overhead of Coordinated Checkpointing




Wall clock times for an application based on coordinated checkpointing for increasing number of nodes
Source: [Varela, Ferreira and Riesen; 2010]

# Other Approaches

## Redundant computing

e.g., [Engelmann et al. 2009, Ferreira et al. 2008]

- Spare nodes are usually required (avoid imbalance)
  $\Rightarrow$ large hardware overhead
- rdmGS integrates redundant computing and (reactive) migration concepts *without extra hardware*

## Algorithm-based fault tolerance (ABFT)

e.g., [Huang & Abraham 1984, Chen & Dongarra 2008, Chen 2011]

- Extend input by checksums, detect and recover from errors
- Usually at a higher level than elementary data aggregation
- Deterministic correction vs. randomized "healing"
- Communication overhead vs. slow-down of convergence

$\rightarrow$ Could complement each other ?

## Current status

- DDAAs: “self-healing” methods
  - $\rightarrow$ Lower overhead
  - $\rightarrow$ Scale asymptotically like parallel reduction
  - $\rightarrow$ Failures slow down convergence
- rdMGS: redundancy on top of DDAAs
- Concept has potential
- Details depend strongly on context parameters
- Performance penalty in practice to be investigated

## Central questions

- Runtime performance comparison ?
- Depends on
  - Topology
  - Failure rates (ratio to total runtime)
  - (Granularity and structure of) Application
  - Checkpointing interval (ratio to total runtime of application)
  - . . .

## Work in progress

- Quantitative performance evaluation and comparison
- Convergence acceleration
  - The more global information (topology, routing, etc.) you can utilize, the faster you can make it !

*Thank you for your attention !*

`http://rlcta.univie.ac.at`