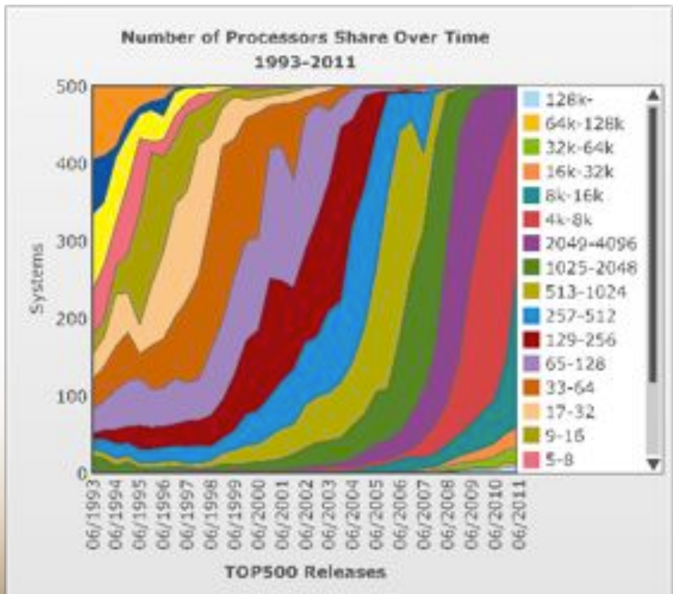
Enabling Software Fault Tolerance with(out) MPI

George Bosilca

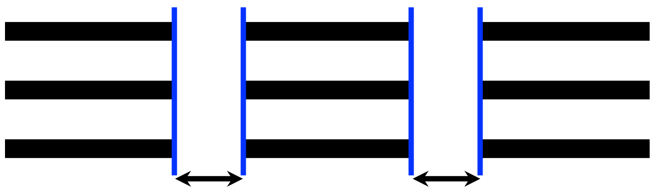
FT: a complex tradeoff



- Transparency
  - **MPI API + ABFT**: MPI returns errors, the application corrects its state online
  - **Application ckpt**: application stores intermediate results, complete restart when failure hits
  - **Automatic**: runtime detects and fix errors

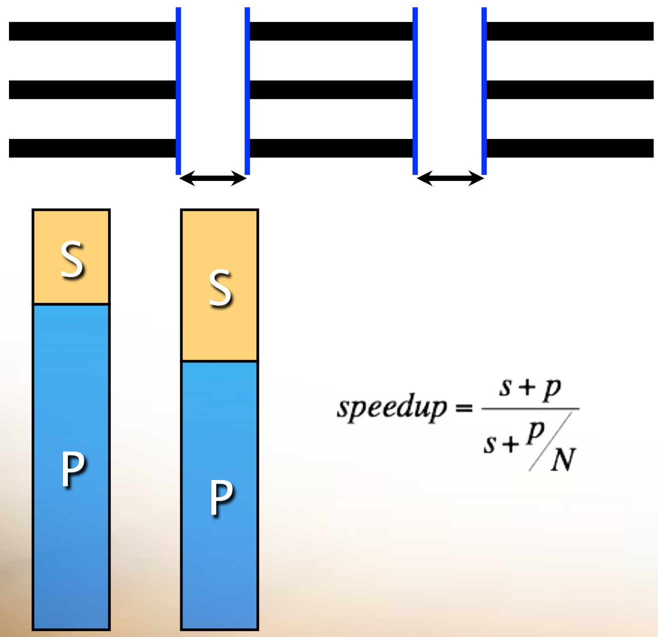
- Checkpoint coordination
  - **Coordinated**: all processes are synchronized, network is flushed before ckpt; all processes rollback to the snapshot
  - **uncoordinated**: each process checkpoints independently; only impacted processes rollback

Coordinated C/R



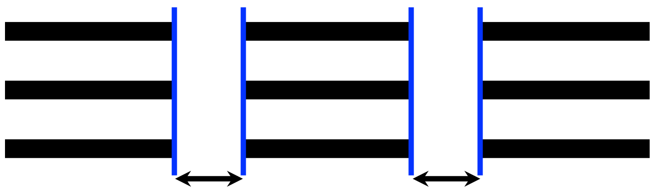
- A complete checkpoint is taken at specified time intervals
- In case of a failure all processes rollback to the last valid checkpoint
- The time to checkpoint strongly depends on the checkpoint support (I/O bandwidth)

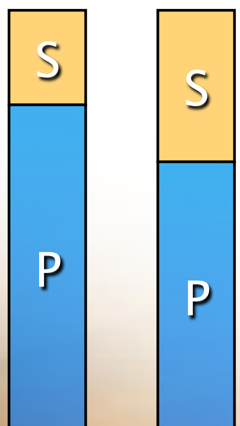
Coordinated C/R



$$speedup = \frac{s+p}{s+p/N}$$

Coordinated C/R





$$speedup = \frac{s+p}{s+p/N}$$

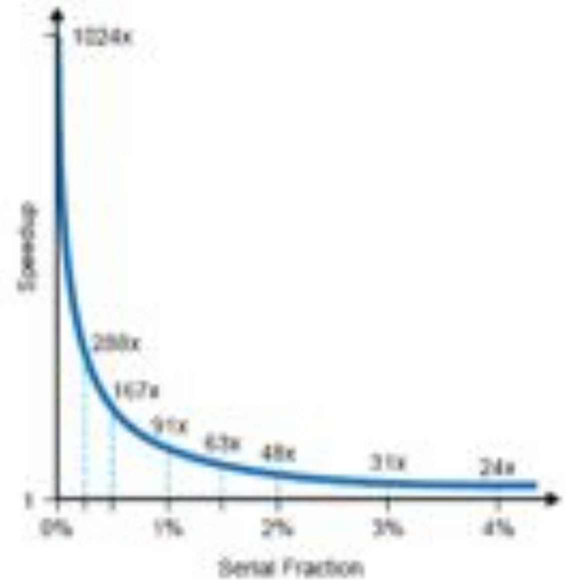
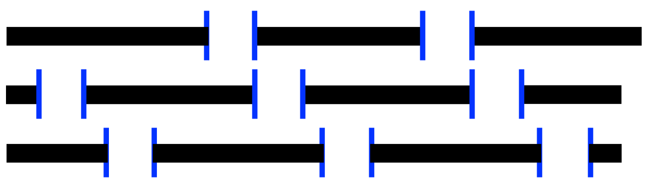
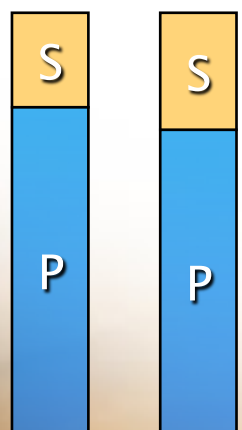


FIGURE 1. Speedup under Amdahl's Law

Uncoordinated C/R

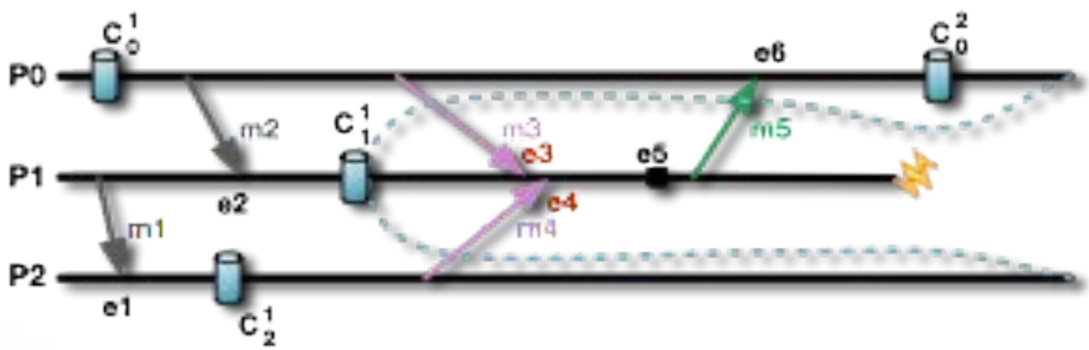




$$speedup = \frac{s+p}{s+p/N}$$

- A single checkpoint is taken at specified time intervals
- In case of a failure one process rollback to the last valid checkpoint
- The time to checkpoint barely depends on the checkpoint support (I/O bandwidth)

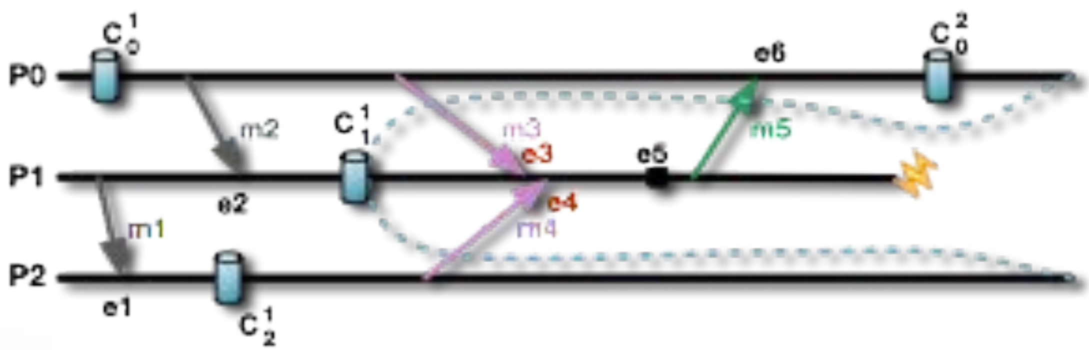
Message Logging



- Grey: past messages
- Dashed: **Recovery line**
- Pink: **in-transit messages**
- Green: **orphan messages**

- **Non-deterministic outcomes** (ordering e_3,e_4 and e_5) are stored (stable storage)**:** **Event Logging**
- **The payload of every message** is stored (volatile memory)**:** **Sender-Based Logging**

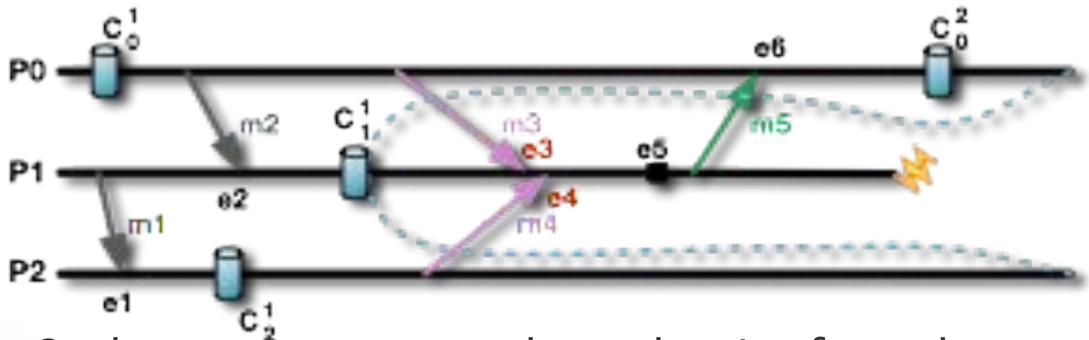
Message Logging



- Special issues
  - Orphan Messages
  - In-transit messages

- Grey: past messages
- Dashed: **Recovery line**
- Pink: **in-transit** messages
- Green: **orphan** messages

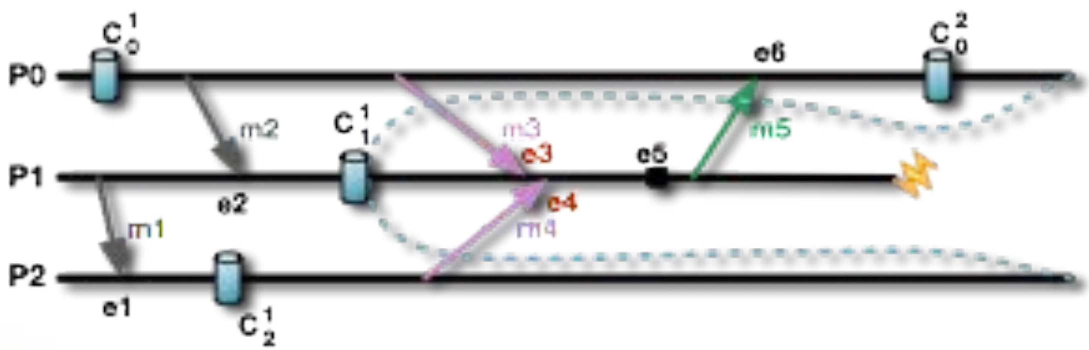
Orphan Messages



- Orphan message carry dependencies from the future to the past, impact of non-deterministic events
- $C^2{}_0$ needs to be discarded! Useless checkpoints, possible domino effect
- **Hence, to form a complete recovery set, the recovery line must adjunct the outcome of non-deterministic events**

- Grey: past messages
- Dashed: **Recovery line**
- Pink: **in-transit messages**
- Green: **orphan messages**

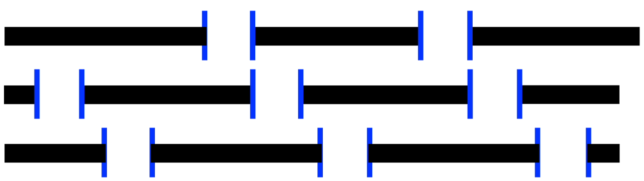
In-transit Messages



- In-transit messages are sent in the past
- If P_0 does not rollback, recovery of P_1 is impossible (missing m_3)
- **Hence to form a complete recovery set, the recovery line must include all in-transit messages**

- Grey: past messages
- Dashed: **Recovery line**
- Pink: **in-transit** messages
- Green: **orphan** messages

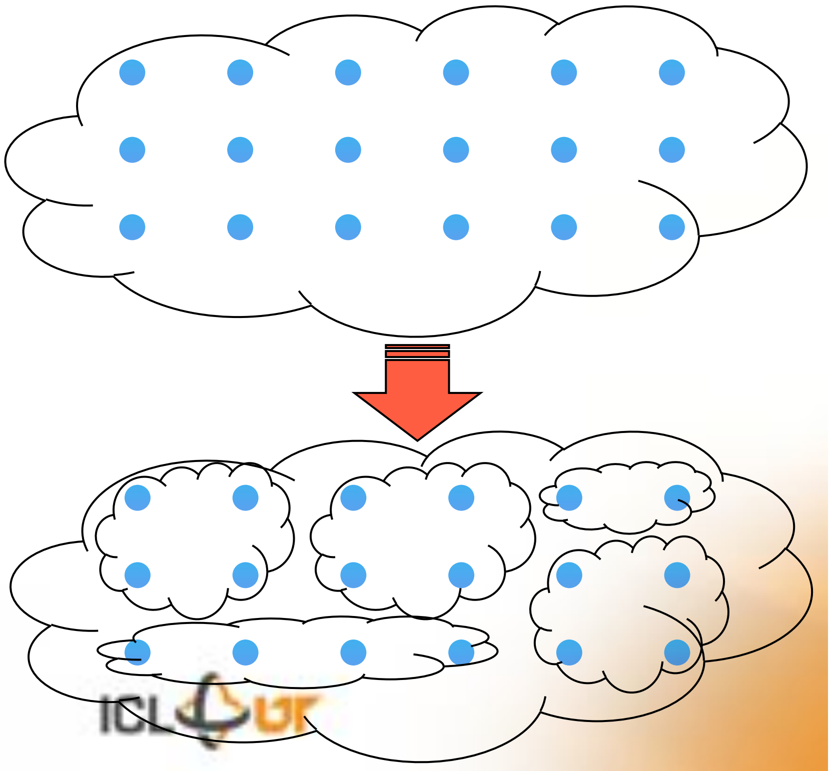
Uncoordinated C/R



- Logging the messages requires memory
  - Potential for decreasing the available memory (direct impact on performance).
- Bounding the logging memory will increase the checkpoint frequency
- It is critical to reduce the amount of logged data

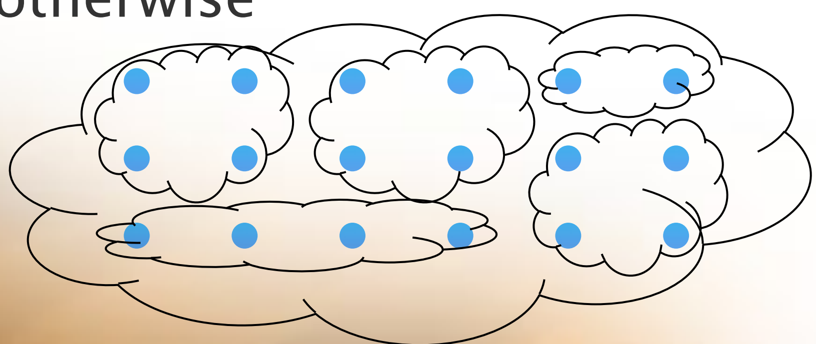
Reducing memory for logging

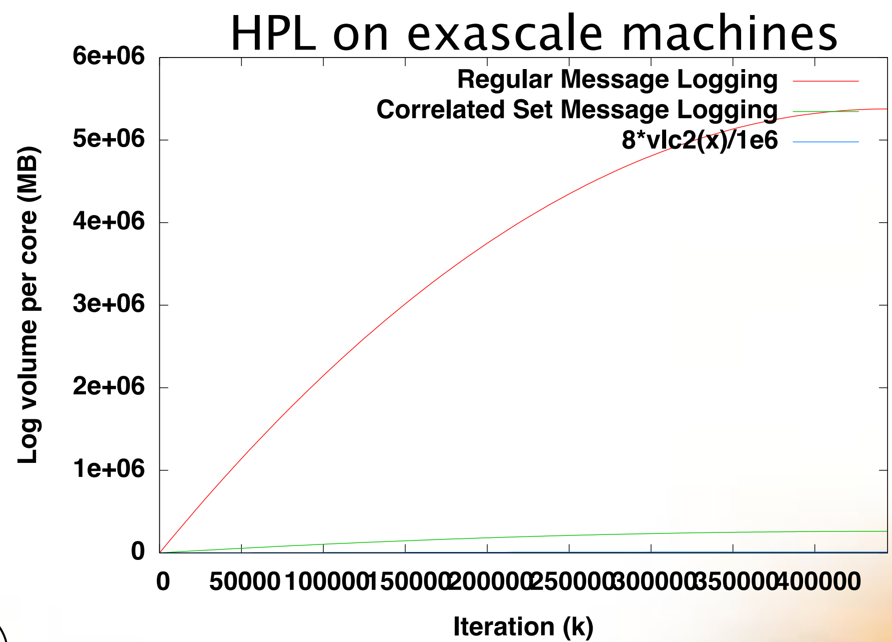
- Correlated sets (or similar)
  - No message logging between peers in the same set
  - Coordinated C/R on a correlated set, uncoordinated otherwise



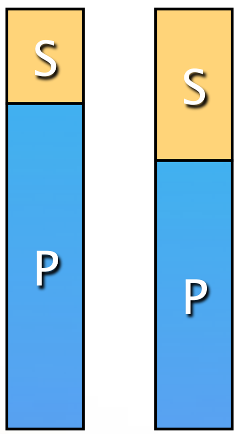
Reducing memory for logging

- Correlated sets
  - No message logging between peers in the same set
  - Coordinated C/R on a correlated set, uncoordinated otherwise





Is the any viable alternative?



$$speedup = \frac{s+p}{s+p/N}$$

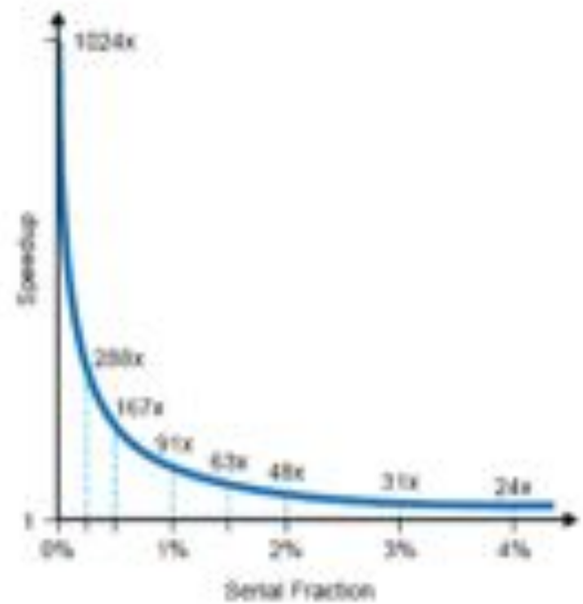


FIGURE 1. Speedup under Amdahl's Law

Algorithmic base fault tolerance

- Deal with the fault deep inside the algorithm
- Win: only save the minimum required data, and only when necessary

AMEND THE MPI STANDARD TO HANDLE FAULTS

Amend the MPI standard: FT-MPI

- Define the behavior of MPI in case an error occurs
- Give the application the possibility to recover from process-failures
- A regular, non fault-tolerant MPI program runs using a FT MPI
- Stick to the MPI-1 and MPI-2 specification as closely as possible (e.g. no unnecessary function calls)
- What FT-MPI does not do:
  - Recover user data (e.g. automatic checkpointing)
  - Provide transparent fault-tolerance

The FT-MPI specification

- General steps when dealing with fault tolerance:
  - Failure detection
  - Notification
  - Recovery procedure
- Questions for the recovery procedure:
  - How to start recovery procedure?
  - What is the status of MPI objects after recovery ?
  - What is the status of ongoing messages after recovery ?


Appendix A
FT-MPI: Proposal for Extentions to the Message-Passing
Interface for Process Fault-Tolerance

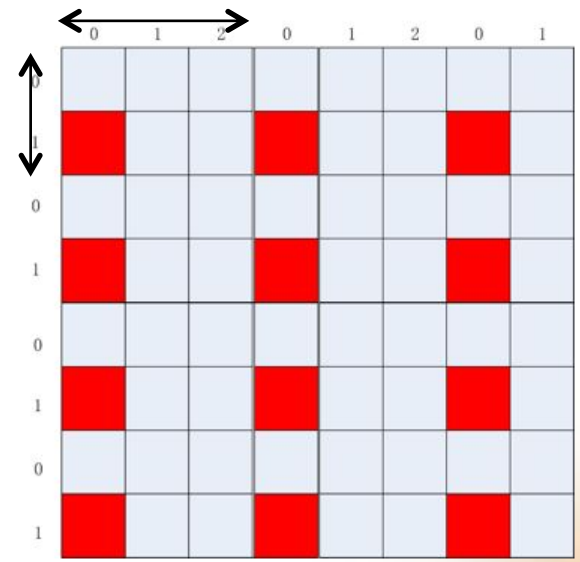
February 16, 2004

Innovative Computing Laboratory,
Computer Science Department,
University of Tennessee, Knoxville

ALGORITHM BASED FAULT-TOLERANCE
QR / LU / CHOLESKY

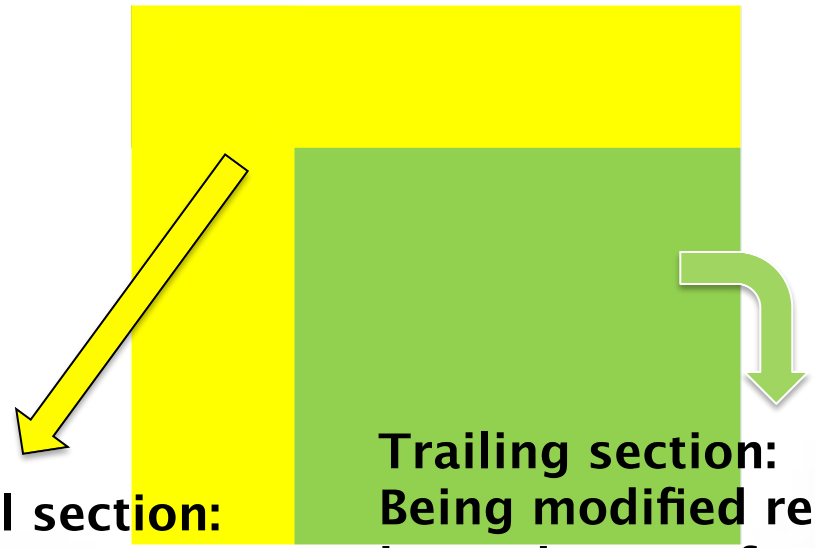
Dense one-sided factorizations

- Scalapack data distribution over PxQ processes (2 x 3)
- When a process dies several pieces of data are lost
- Recovering the dead process means recovering all the lost data



Dense one-sided factorizations

- Such approach can be used for multiple purposes such as hard and soft errors



Panel section

- The usual suspect: checkpoint the useful data

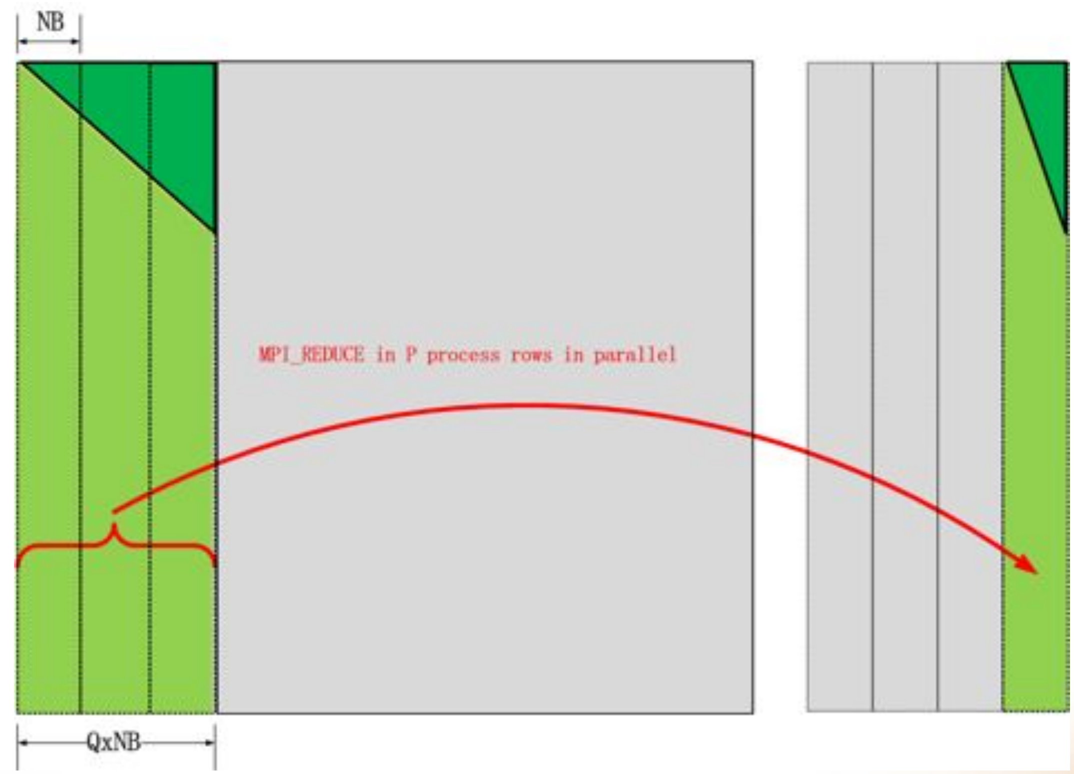
$$C = \alpha_1 C_1 + \alpha_2 C_2 + \cdots + \alpha_{n-1} C_{n-1} + \alpha_n C_n$$

$$\alpha_2 C_2 = C - (\alpha_1 C_1 + \alpha_3 C_3 + \cdots + \alpha_{n-1} C_{n-1} + \alpha_n C_n)$$

- Requirements:
  - Low performance impact to host algorithm
  - Global vertical checkpointing is NOT scalable

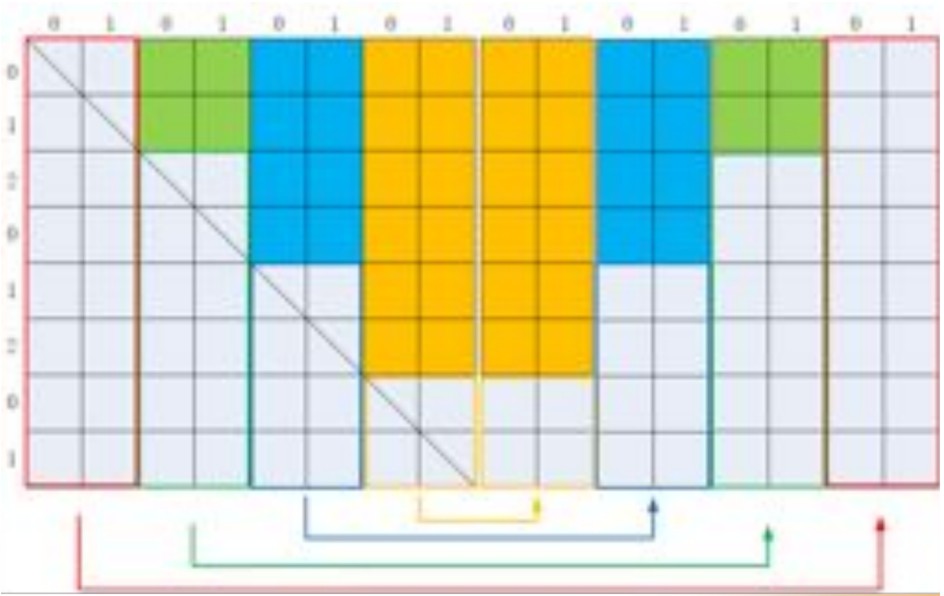
Panel section

- Checkpoint every Q panels to cope with data distribution
- Store the checkpointed data reversly, to minimize the GEMM cost.

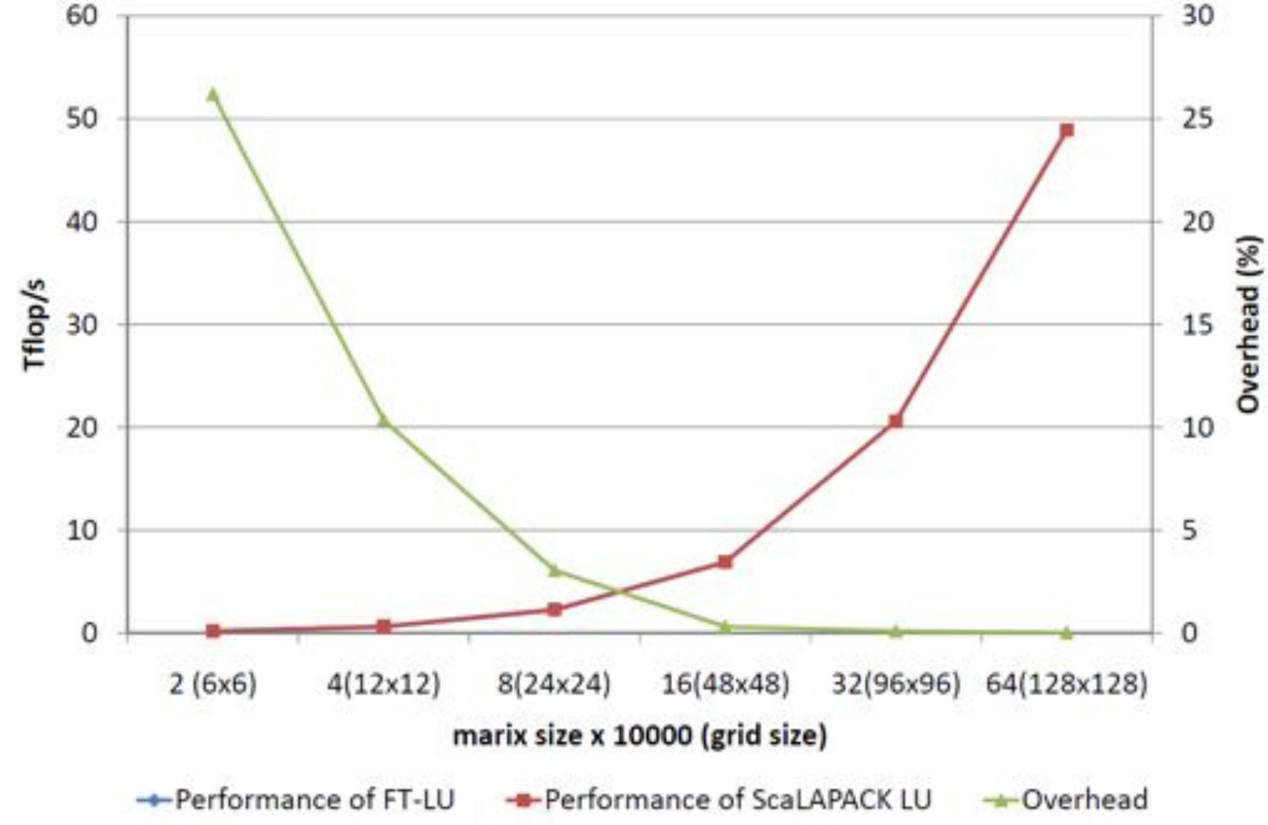


Recovery Algorithm

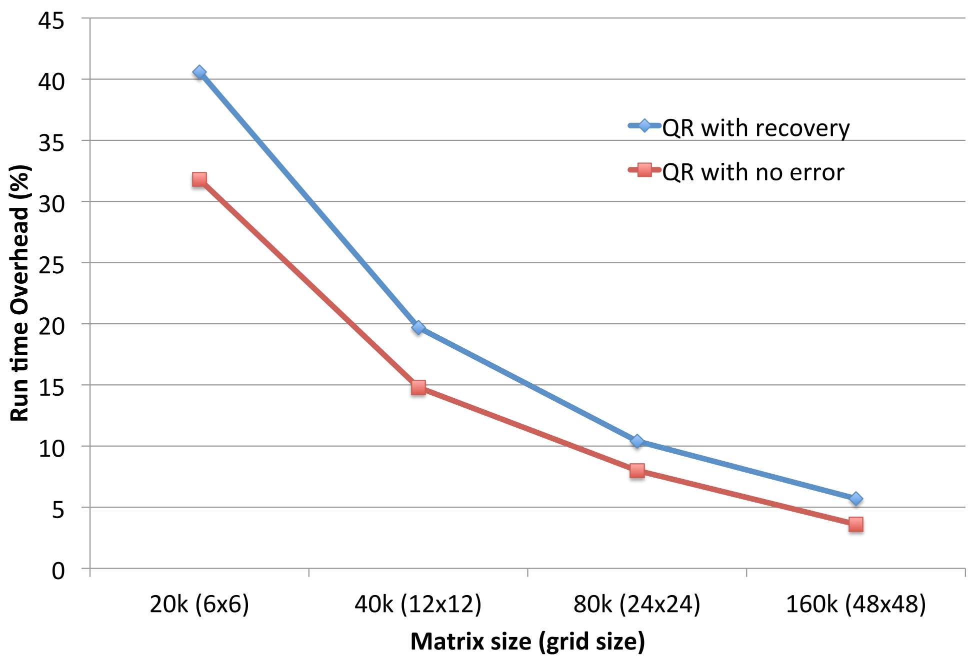
- Recover the MPI process by the [FT MPI approaches]
- Reverse the checkpointing process to retrieve the data back
- Panel section uses local copy to roll back and redo up to the last panel factorization



Performance (Hard error) – LU



Performance (Hard error) – QR



Weak scalability on FT-QR: overhead with failures

Performance (Soft error)

- Same approach can be used to solve the soft errors issues
  - With no additional cost

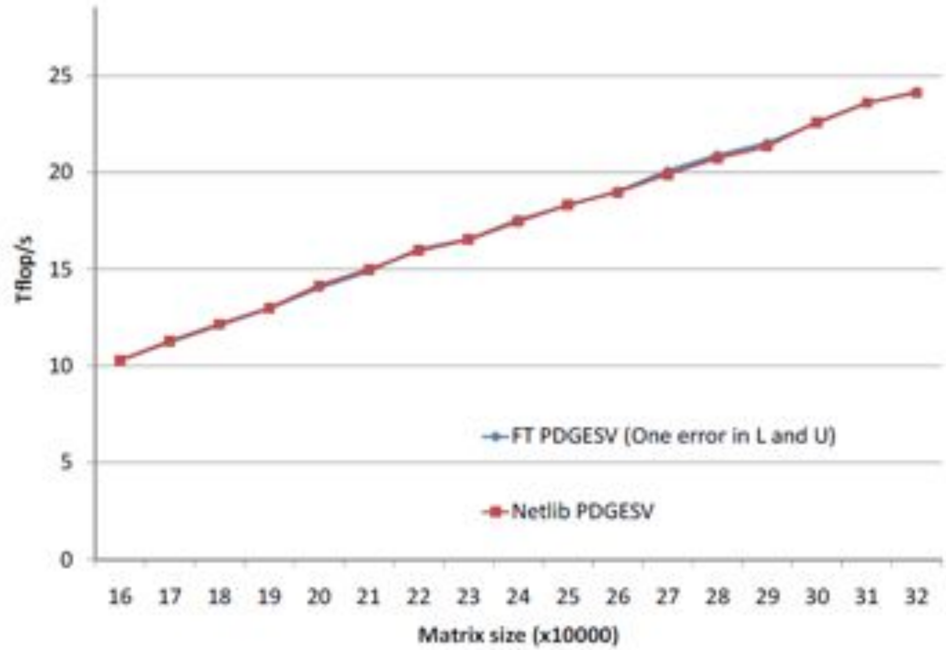
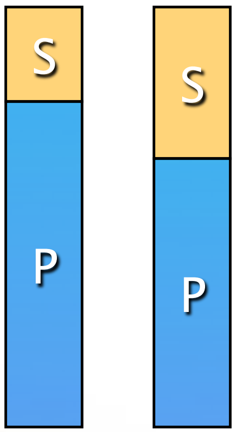


Fig. 13: PDGESV performance with and without soft error resilience on 24576 cores of Cray XT5.

Are we there yet?



$$speedup = \frac{s + p}{s + p/N}$$

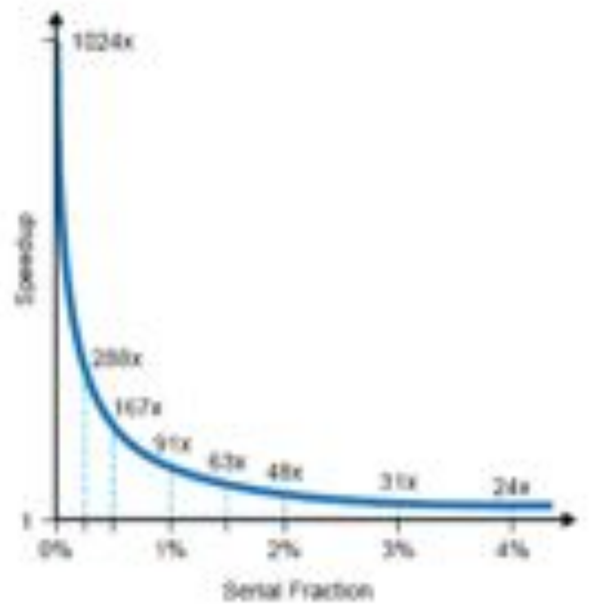


FIGURE 1. Speedup under Amdahl's Law

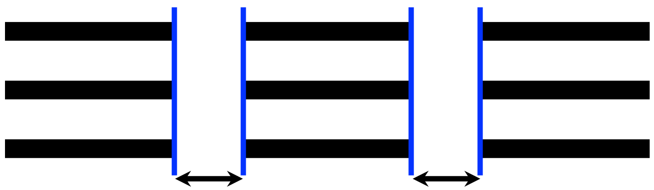
Algorithmic base fault tolerance requires drastic modification of the applications as well as of the MPI standard.

Scale back the MPI requirements and merge several approaches together

MPI and Error handling

- MPI_ERRORS_ARE_FATAL (Default mode):
  - Abort the application on the first error
- MPI_ERRORS_RETURN:
  - Return error-code to user
  - State of MPI undefined
  - "…does *not* necessarily allow the user to continue to use MPI after an error is detected. The purpose of these error handler is to allow a user to issue user-defined error messages and take actions unrelated to MPI…An MPI implementation is free to allow MPI to continue after an error…"(MPI-1.1, page 195)
  - "*Advice to implementors*: A good quality implementation will, to the greatest possible extent, circumvent the impact of an error, so that normal processing can continue after an error handler was invoked."

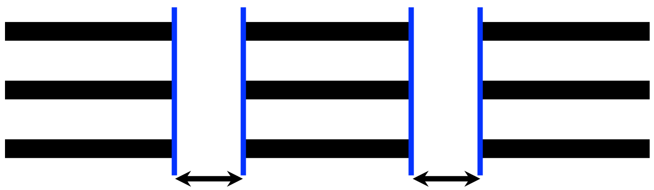
[Coordinated] C/R



- A complete checkpoint is taken only when required
  - we have the optimum checkpoint interval

- A complete checkpoint is taken at specified time intervals
- In case of a failure all processes rollback to the last valid checkpoint
- The time to checkpoint strongly depends on the checkpoint support (I/O bandwidth)

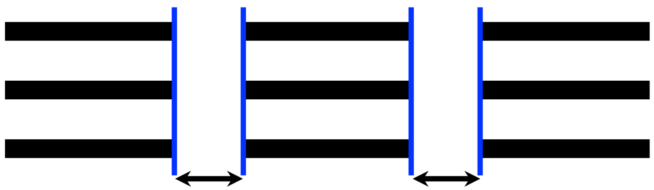
[Coordinated] C/R



- All except the deal processes

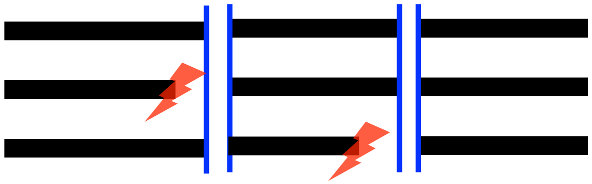
- A complete checkpoint is taken at specified time intervals
- In case of a failure all processes rollback to the last valid checkpoint
- The time to checkpoint strongly depends on the checkpoint support (I/O bandwidth)

[Coordinated] C/R



- The checkpoint can happen locally as long as the next allocation cover the same resources.
- A complete checkpoint is taken at specified time intervals
- In case of a failure all processes rollback to the last valid checkpoint
- The time to checkpoint strongly depends on the checkpoint support (I/O bandwidth)

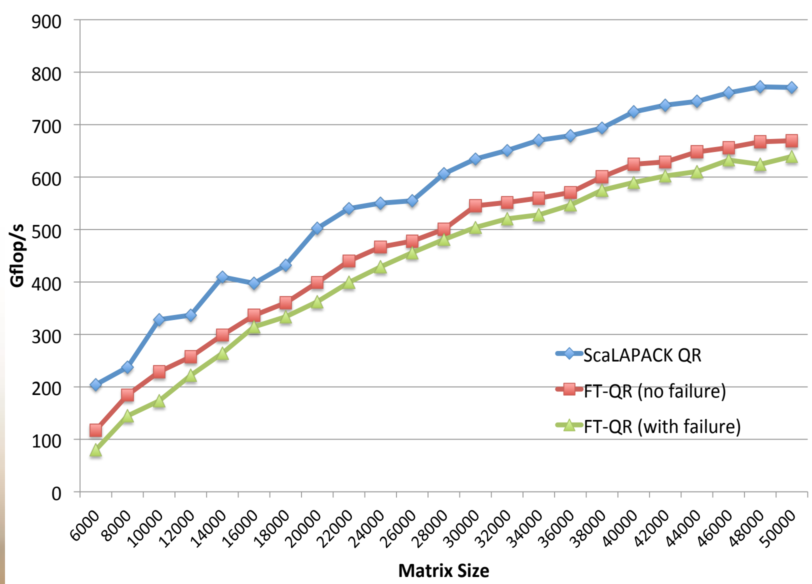
On-demand C/R



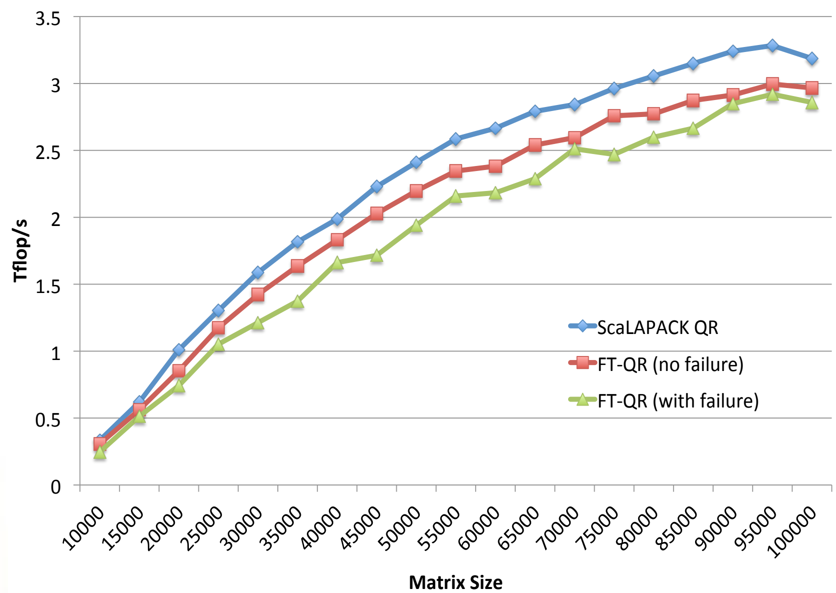
- Checkpoint all remaining processes when a fault is detected
  - Minimal fault-free overhead (identical to ABFT)
- Checkpoint can happen locally, as long as the next allocation cover the same resources.

On-demand C/R

Dancer (16x8)



Kraken (24x24)



Conclusions

- There are ways to recover even for difficult algorithms without hindering performance or scalability
- Hybrid solutions seems to provide simpler systems (with decent performance penalties)
- How to efficiently compose fault tolerant approaches (deeper than just at the runtime level)