A fault tolerant application using the GASPI communication layer

Faisal Shahzad[1], Markus Wittmann[1], Moritz Kreutzer[1], Rui Machado[2], Andreas Pieper[3], Thomas Zeiser[1], Georg Hager[1], Gerhard Wellein[4]

Motivation

Tianhe-2
Perf.: 33.8 PFlop/s
core count: 3,120,000

Sequoia
Perf.: 17.17 PFlop/s
core count: 1,572,864
Mean Time To Failure[1] = 19hr

Large scale applications must expect failures in post petascale machines

MPI still does not provide a solution to a single process failure.

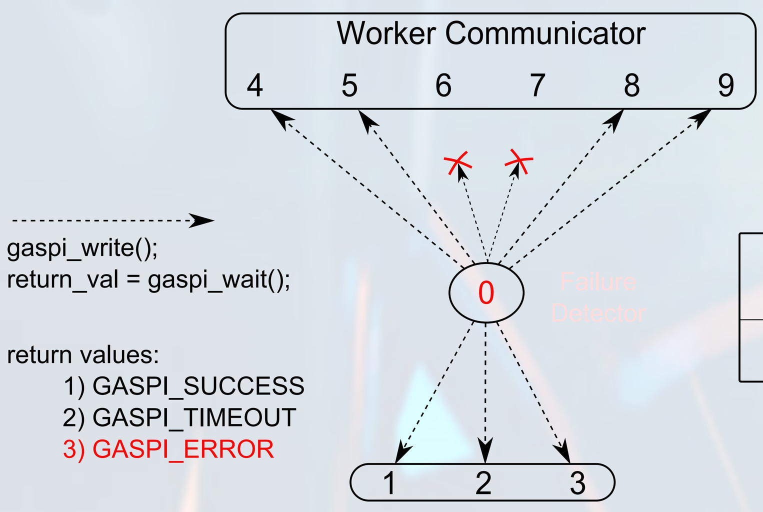
Introduction & Methodology

GASPI

- GASPI[2] enables fault tolerance via timeout based communication routines.
- A process local health-state vector is updated after every communication call.
- A process is considered as a failed process if it is unable to respond to a communication request within a certain amount of time.
- Health state of a process gets refreshed after every successful/ unsuccessful communication.
- In order to have a consistent view of all processes' health, a process must communicate with every other process.

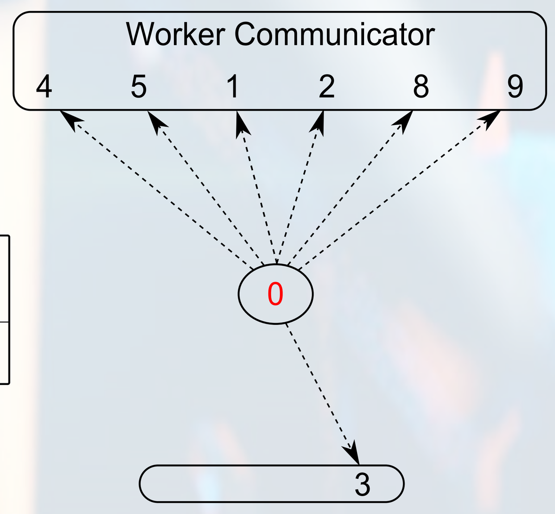
Health Check

- Program started with 'x' redundant processes.
- One of the redundant processes also acts as fault detector.
- Health check via one sided ping.

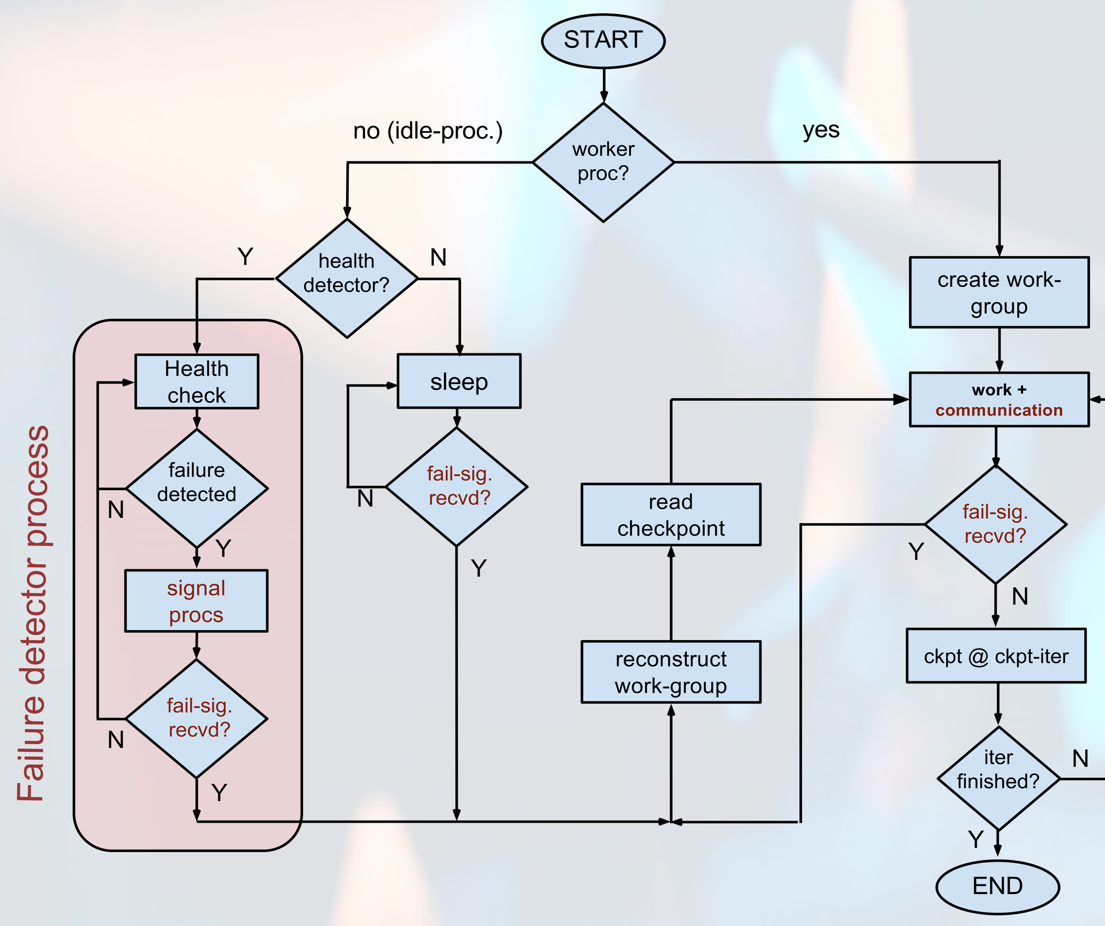


Failed proc(s) IDs	Rescue proc(s) IDs
6, 7	1, 2

Detector processes informs every process about failure details via gaspi_write()



Program Flow



Benchmark & Results

LANCZOS Algorihtm

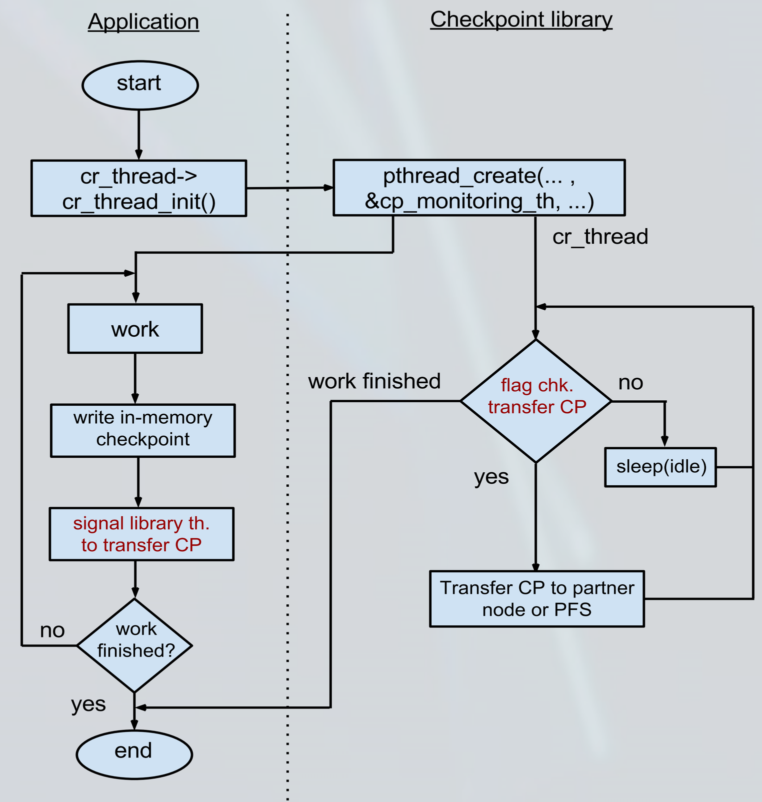
- Prototype for Krylov subspace method
- Eigenvalue computation

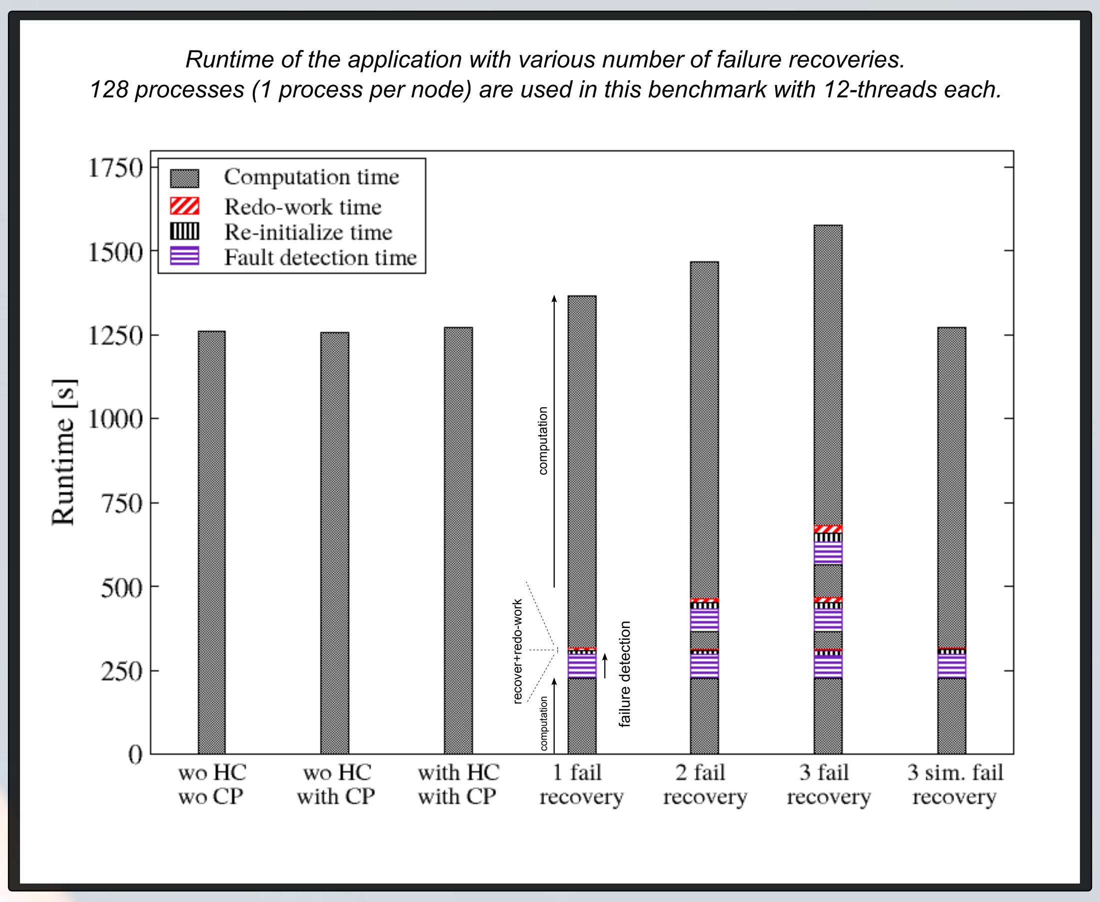
for i:=1,2, ..., ConvergenceCriterion **do**
  function LANCZOS-STEP
    $\omega_j \leftarrow A\nu_j$
    $\alpha_j \leftarrow \omega_j.\nu_j$
    $\omega_j \leftarrow \omega_j - \alpha_j\nu_j - \beta_j\nu_{j-1}$
    $\beta_{j+1} \leftarrow \|\omega_j\|$
    $\nu_{j+1} \leftarrow \omega_j/\beta_{j+1}$
  end function
  $CalcMinimumEigenVal()$
end for

Checkpoint Data Structure

- Each process once stores matrix communication data structure (to be later used by rescue process in case of a failure).
- Two recent Lanczos vectors are stored at each checkpoint iteration with recently computed eigenvalues.
- Multi-level checkpointing via asynchronous library thread:

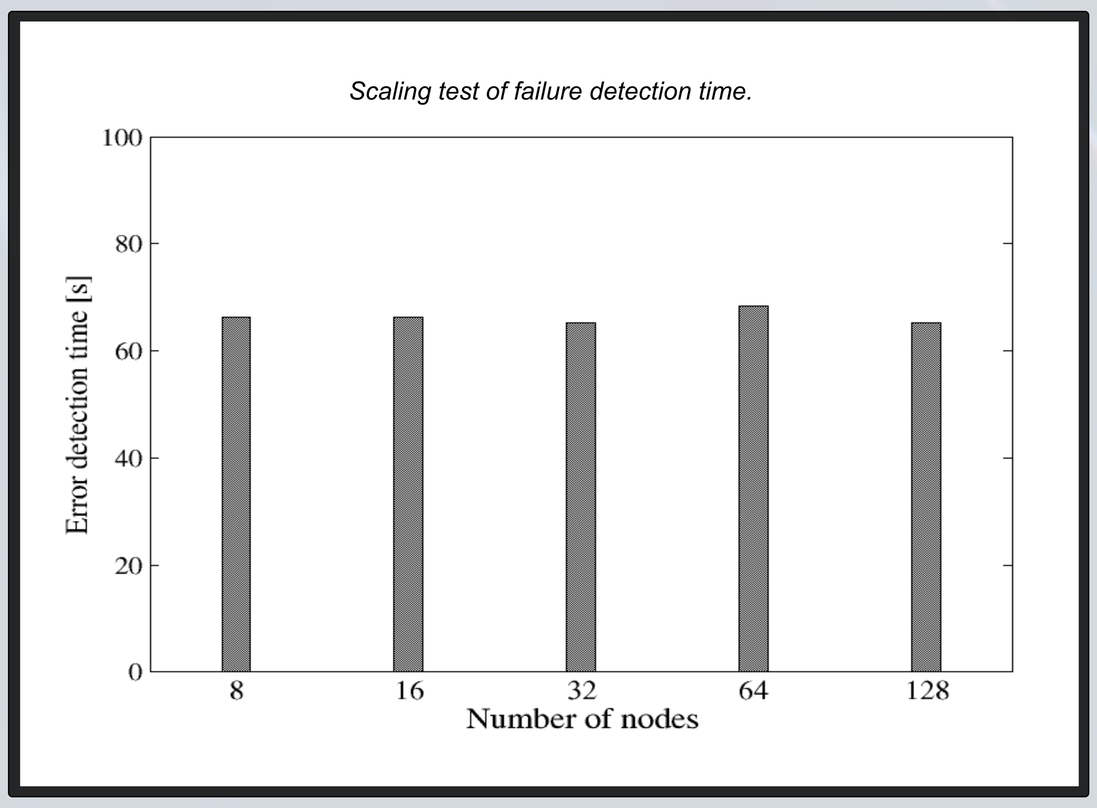
1) Node level
2) Neighbor node level
3) Parallel file system level





Overhead component	Time (seconds)
Failure detection time (OH_1)	65.3
Rebuild communicator + read checkpoint time* (OH_2)	12.8
Redo-work** (OH_3)	9.6

Breakdown of the overhead time for process 0 during failure recovery.
* application dependent, ** failure instance dependent



Conclusion & Future Work

- Worker processes are not interupted for health checking purpose.
- Overhead only in case of worker failure(s).
- Scalable health check approach.
- Redo-work after failure recover <=> Checkpoint frequency.

References

1. Jack Dongarra. Emerging Heterogeneous Technologies for High Performance Computing. Invited talk. website: http://www.netlib.org/utk/people/JackDongarra/SLIDES/hcw-0513.pdf, IPDPS'13, May 2013.
2. GASPI project website: http://www.gaspi.de/en/project.html

1) Erlangen Regional Computing Center (RRZE), University of Erlangen-Nuremberg, Germany
2) Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany
3) Institute of Physics, University of Greifswald, Greifswald, Germany
4) Department of Computer Science, University of Erlangen-Nuremberg, Germany
Email: faisal.shahzad@fau.de

Funded by: ESSEX SPPEXA