ERLANGEN REGIONAL
COMPUTING CENTER

Application driven fault tolerance and asynchronous checkpointing

Faisal Shahzad

02.03.2015

Partially funded by BMBF project FeTol

HPC-Software für skalierbare Parallelrechner
gefördert durch das
Bundesministerium für Bildung und Forschung

Partially funded by DFG Priority Programme1648

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Challenge

- Nowadays, the increasing computational capacity is mainly due to extreme level of hardware parallelism.
- The reliability of hardware components does not increase with the similar rate.
- With future machines, the Mean time to failure is expected to be in minutes and hours.
- Absence of fault tolerant environment will put precious data at risk.

Checkpoint/Restart optimizations

1. Application level checkpointing
   - Minimal checkpoint data

2. **Asynchronous checkpointing**

3. **Multi-level checkpointing (PFS/remote node/localFS)**

Hide / avoid costs of computational costs of checkpoints

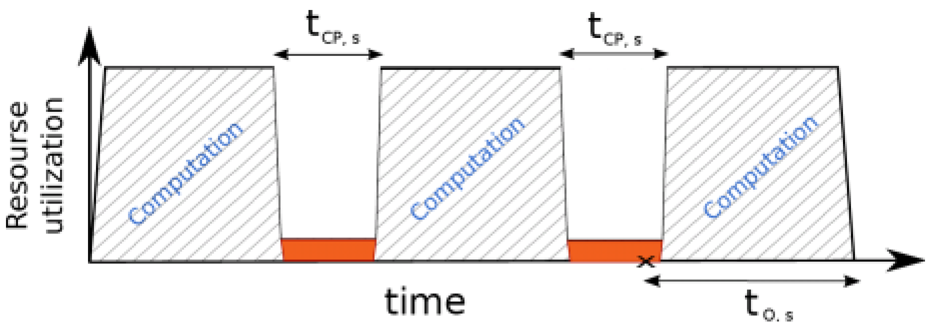
4. Checkpoint compression

5. …

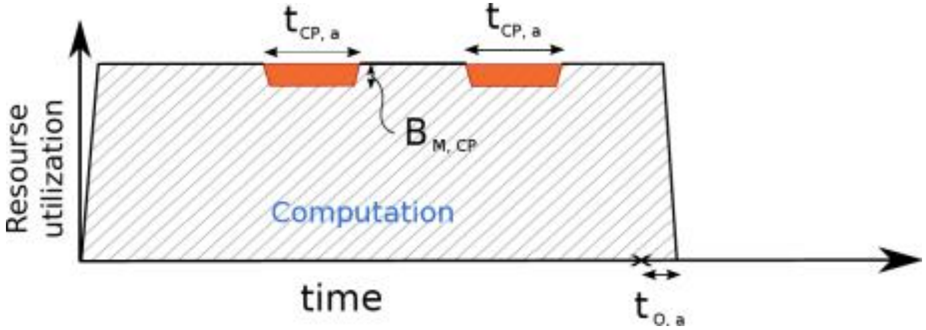
ASYNCHRONOUS CHECKPOINTING

Synchronous vs. asynchronous checkpointing

- Synchronous checkpointing:
  - Computation halts for I/O time
  - High execution time overhead



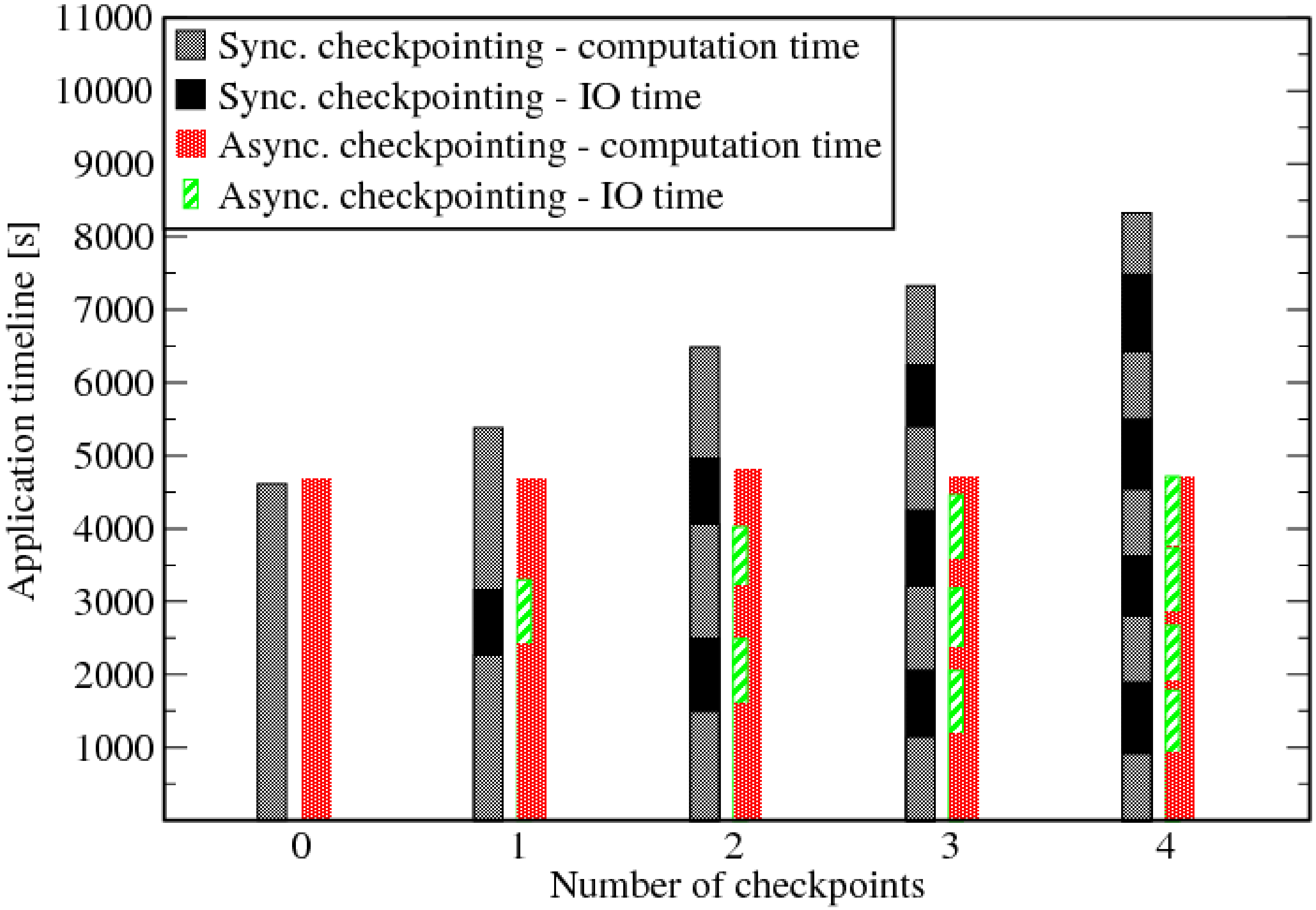
- Asynchronous checkpointing:
  - Using dedicated threads for performing asynchronous I/O
  - Low execution time overhead
  - An in-memory copy of checkpoint is required.



Asynchronous vs. Synchronous Checkpointing

- Benchmark (LiMa)

Num. of nodes = 128, np = 1536, PFS = LXFS, Aggregated CP size = 800GB/CP

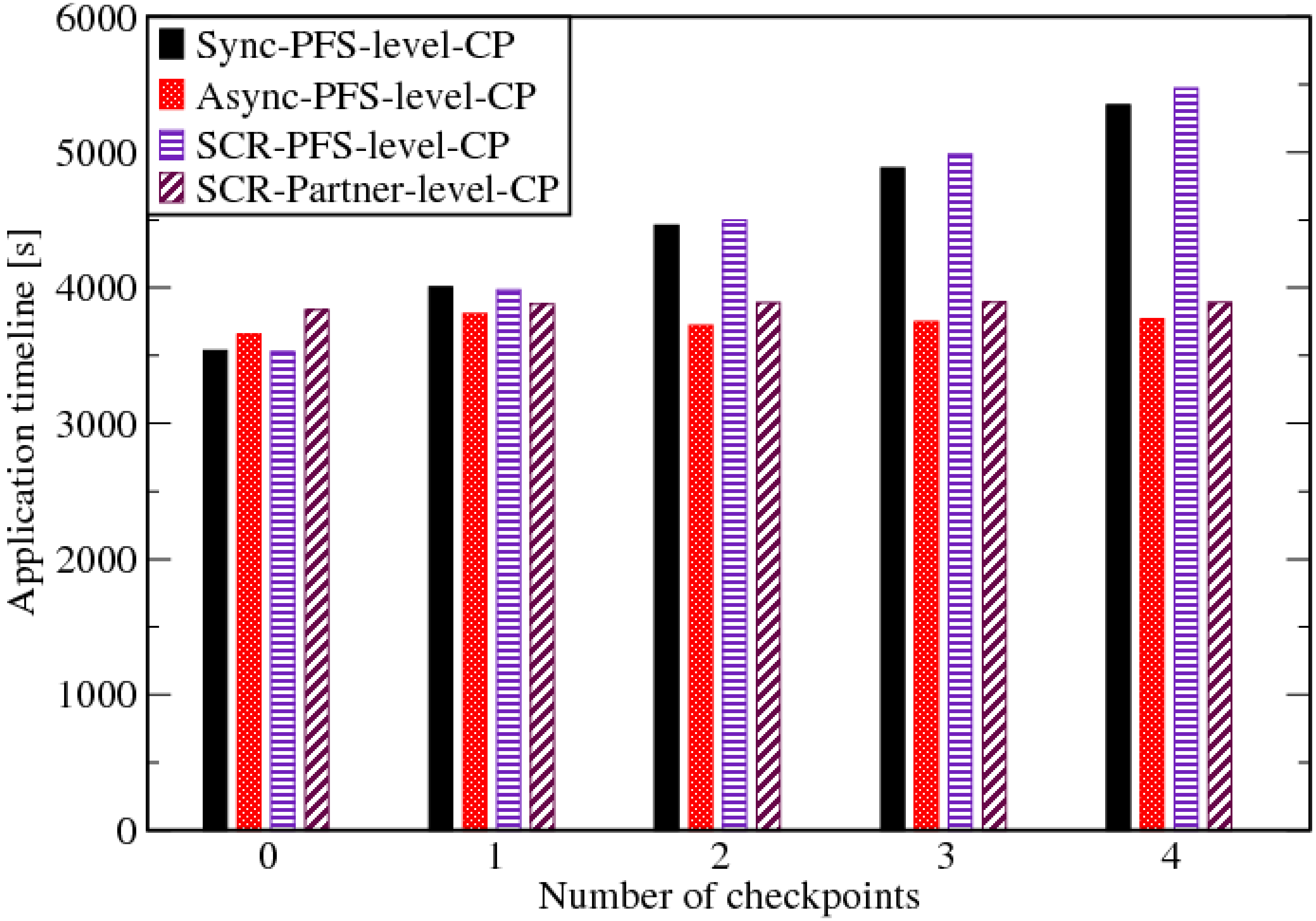


% overhead

- **1 Sync. CP = 20 %**
- **1 Async. CP = 0.4 %**

Async. vs. Sync. vs. SCR Checkpointing

- Benchmark (LiMa)



Num. of nodes = 128, PFS = LXFS,

CP size = 510 GB /CP

- **<u>% overhead:</u>**
- **1 Sync. CP = 13 %**
- **1 Async. CP = 1.3 %**
- **1 Partner. CP = 1 %**

SCR: A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, “Design, Modeling, and Evaluation of a Scalable Multilevel Checkpointing System,” in Proceedings of the 2010 ACM/IEEE International Conference for HPC, Networking, Storage and Analysis, Washington,DC, USA

Remarks: Asynchronous checkpointing

- Effective implementation of C/R and effective resource utilization can reduce overhead to minimum level.
- The overhead due to I/O bottlenecks can be reduced with asynchronous checkpointing approach.

- Critical parameter → checkpoint frequency
  - System parameters, checkpoint latency, restart time ,…
  - Upper limit on the number of checkpoints

- Limitations
  - In-memory copy of the checkpoint data costs
    1. Extra memory space (in worst case, can be up to 50%)
    2. Time (can be avoided)

AUTOMATIC FAULT TOLERANCE APPLICATION (AFT) WITH GPI

Automatic Fault Tolerance Application (AFT)

- **Automatic fault tolerance application (AFT)**
  - In the absence of failed processes, the algorithm itself is able to detect and correct the incorrectly produced results
- **Fault Tolerant - MPI ?**
- **GPI (Global address space Programming Interface)**
  - Fault tolerance → In case of single node failure, rest of the nodes stay up and running

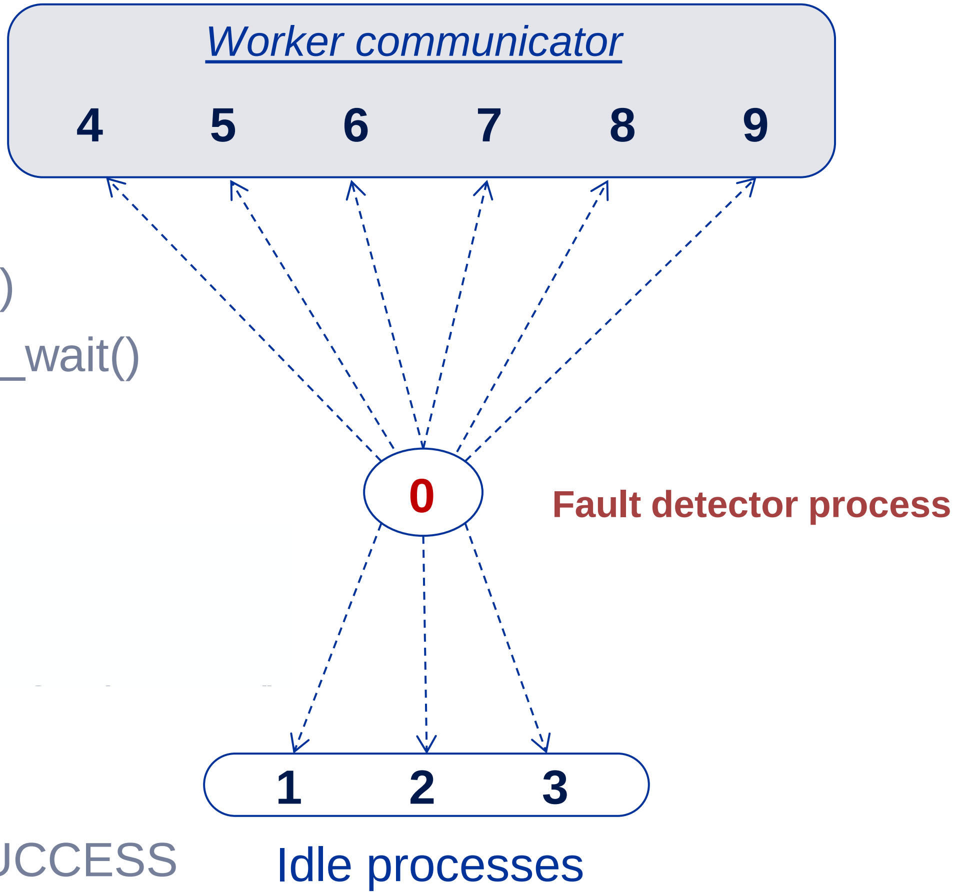
AFT: GPI Introduction

- Developed by Fraunhofer IWTM
- Based on PGAS programming model
- Two memory parts
  - Local: only local to the GPI process (and its threads)
  - Global: Available to other processes for reading and writing.
- Enables fault tolerance
  - Provides TIMEOUT for every communication call.
  - Each process maintains a health vector with the communicating partners.

Failure detector:

gaspi_write()

return_val = gaspi_wait()



return_val:

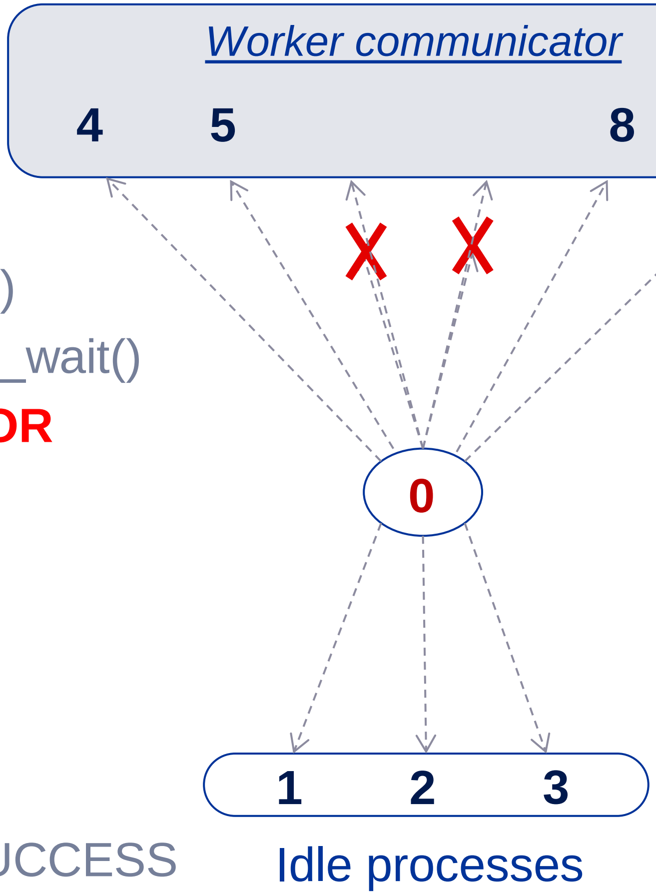
1) GASPI_SUCCESS
2) GASPI_TIMEOUT
3) GASPI_ERROR

Failure detector:

gaspi_write()

return_val = gaspi_wait()

GASPI_ERROR



Failure detector process

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

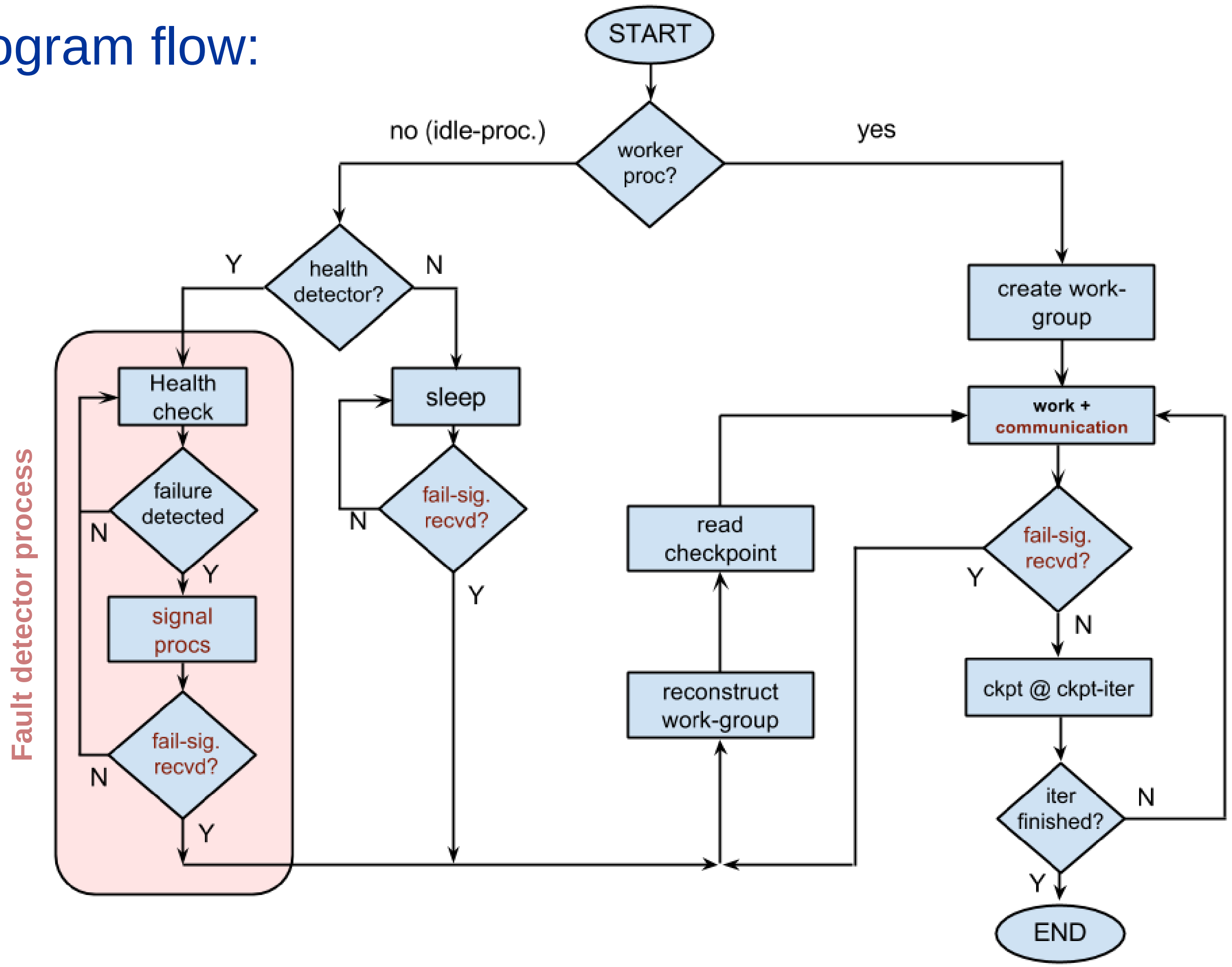
- Detector processes informs every process about failure details via gaspi_write().

return_val:

1) GASPI_SUCCESS
2) GASPI_TIMEOUT
3) GASPI_ERROR

Automatic Fault Tolerance Application

- Program flow:



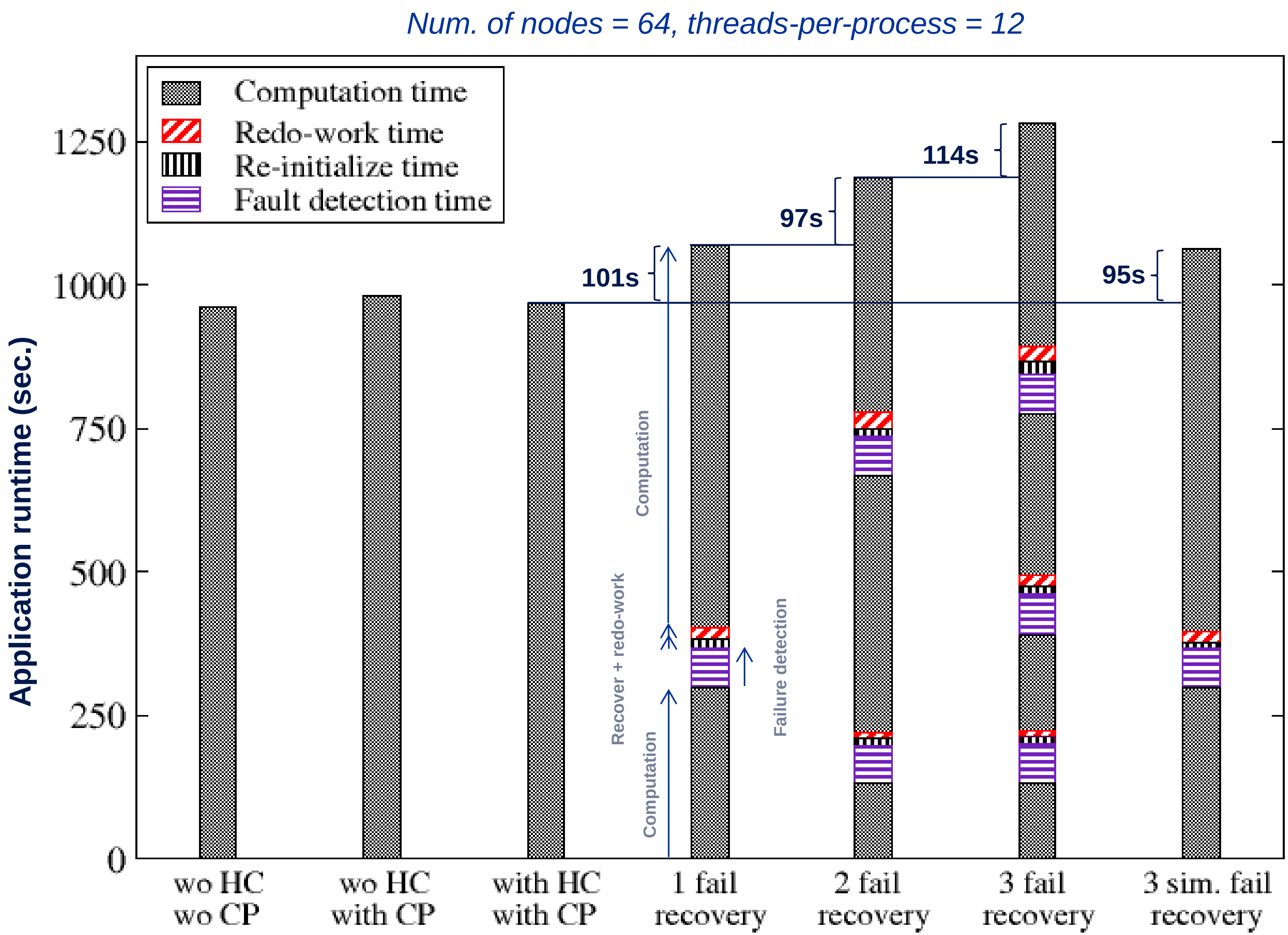
Benchmarks: Test bed

- Lanczos algorithm:

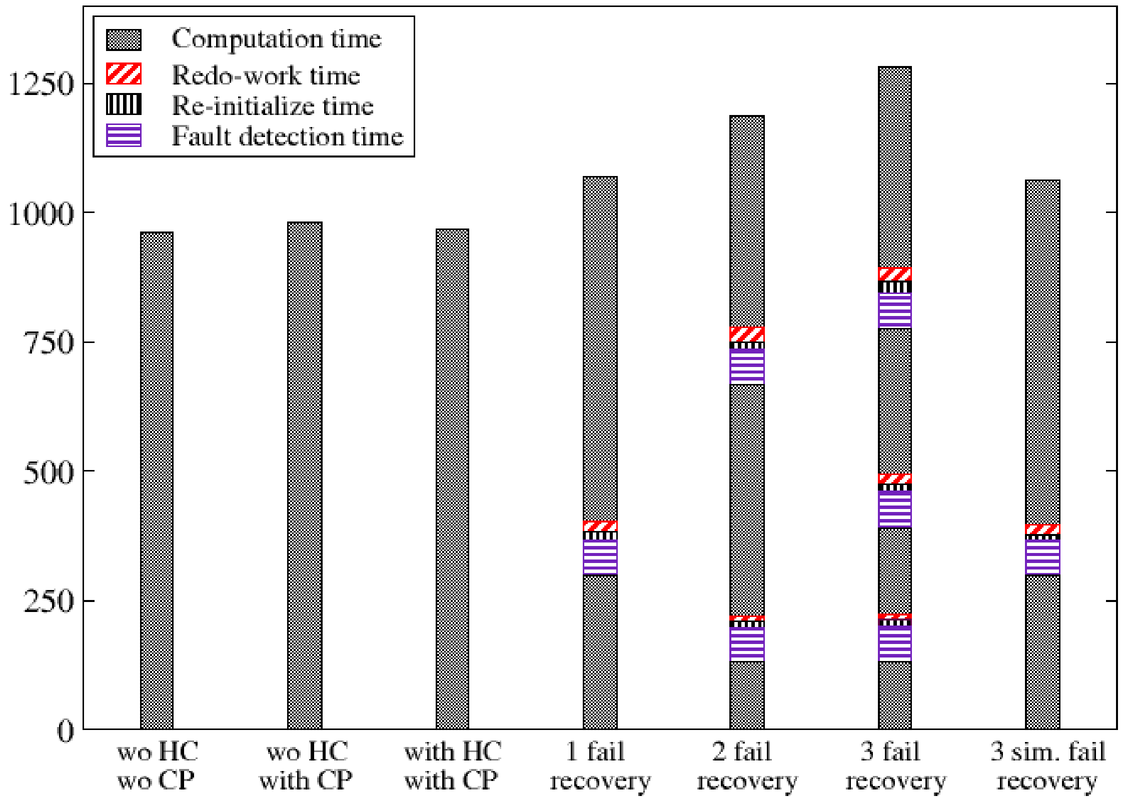
for i:=1,2, ..., ConvergenceCriterion **do**
  function LANCZOS-STEP
    $\omega_j \leftarrow A\nu_j$
    $\alpha_j \leftarrow \omega_j.\nu_j$
    $\omega_j \leftarrow \omega_j - \alpha_j\nu_j - \beta_j\nu_{j-1}$
    $\beta_{j+1} \leftarrow \|\omega_j\|$
    $\nu_{j+1} \leftarrow \omega_j/\beta_{j+1}$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

- Checkpoint data structure:
  - After startup: Every process once stores matrix communication data structure.
  - Two recent Lanczos vectors are stored at each checkpoint iteration.
  - Recently calculated eigenvalues.
- Test cluster:
  - LiMa – RRZE, Erlangen

Benchmark:


Num. of nodes = 64, threads-per-process = 12
Computation time
Redo-work time
Re-initialize time
Fault detection time
Application runtime (sec.)
0
250
500
750
1000
1250
101s
97s
114s
95s
Computation
Recover + redo-work
Computation
Failure detection
wo HC wo CP
wo HC with CP
with HC with CP
1 fail recovery
2 fail recovery
3 fail recovery
3 sim. fail recovery

Benchmark:



- Avg. fault detection time (by gaspi_wait): 67 sec.
- Avg. re-initialize time: 16 sec.
- Avg. failure recovery time (without redo-work): 83 sec.
- Redo work: dependent on instant of failure between 2 checkpoints

Remarks:

- Worker processes remain undisturbed in failure-free application run.
- Overhead only in case of worker failure(s).
- Scalable.
- Redo-Work after failure recovery ⇔ Checkpoint Frequency.

- MPI-ULFM:
  - On going work by MPI Forum's fault tolerance working group to incorporate FT features in MPI-4.
  - Prototype implementation in form of User Level Failure Mitigation (ULFM).

Thank you!

Questions?

Partially funded by DFG Priority Programme1648

SPPEXA

ESSEX

Partially funded by BMBF project FeTol

HPC-Software für skalierbare Parallelrechner
gefördert durch das
Bundesministerium für Bildung und Forschung

ASYNCHRONOUS CHECKPOINTING IN GHOST (ESSEX)

Equipping Sparse Scalable Solvers for Exascale (ESSEX)

Hardware

Fault tolerance
Energy efficiency
New levels of parallelism

Quantum Physics Applications

Extremely large sparse matrices:
eigenvalues, spectral properties,
time evolution

ESSEX

Exascale Sparse Solver Repository (ESSR)

- FT concepts, programming for extreme parallelism
- Sparse eigensolvers, preconditioners, spectral methods
- Quantum physics / chemistry
- ESSEX applications: Graphene, topological insulators, …

Basic building blocks library: GHOST

General, Hybrid and Optimized Sparse Toolkit

GHOST

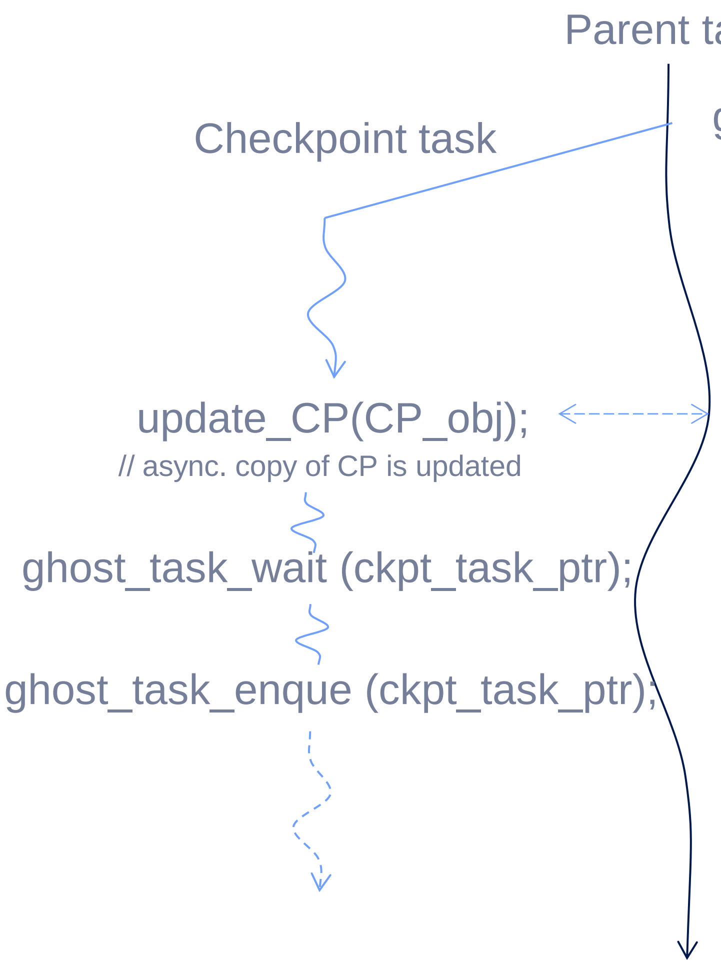
- Basic tailored sparse matrix / vector operations
- CRS or **SELL-C-σ*** (**unified format**) storage schemes
- (Block-)SpMVM: SIMD intrinsic (AVX, SSE, MIC) & CUDA kernels
- Dense vector /matrices: row-/column-major storage

- **Supports** data & **task parallelism** (up to application level)
- MPI + OpenMP + **tasks for concurrent execution**
- **Generic and hardware-aware task management**

- **Application layer triggered checkpoint / restart**
- **Asynchronous checkpointing via tasks**
- **Various checkpoint locations (node, filesystem)**

*****M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop: *A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units.* SIAM Journal on Scientific Computing **36**(5), C401–C423 (2014).

Asynchronous checkpoints via GHOST-task thread:



CP_obj:

- object of ckpt_t type
- ckpt_t class is defined by programmer
- checkpoint object contains the aynchronous copy of the checkpoint

CP_func:

- This function takes an updated copz of CP_obj as argument and writes to PFS..