ERLANGEN REGIONAL
COMPUTING CENTER

Application level asynchronous check-pointing / restart: first experiences with GPI

Faisal Shahzad and Gerhard Wellein

Dagstuhl Seminar „Resilience in Exascale Computing"

September 29, 2014

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

Background

Erlangen Regional Computing Center:

- Tier-2 center in Germany
- Operates compute clusters (200,…,600 nodes)
- Scientist from University of Erlangen and northern Bavaria
- Strong application support group (collaboration with LRZ Munich)

- HPC research focus:
  - Node level performance engineering
  - Hardware efficiency of sparse linear algebra, lattice Boltzmann solvers, stencil computations
  - Hybrid/new programming parallel models

- Leading PI of ESSEX project from SPPEXA

Equipping Sparse Scalable Solvers for Exascale (ESSEX)

Hardware

Fault tolerance
Energy efficiency
New levels of parallelism

Quantum Physics Applications

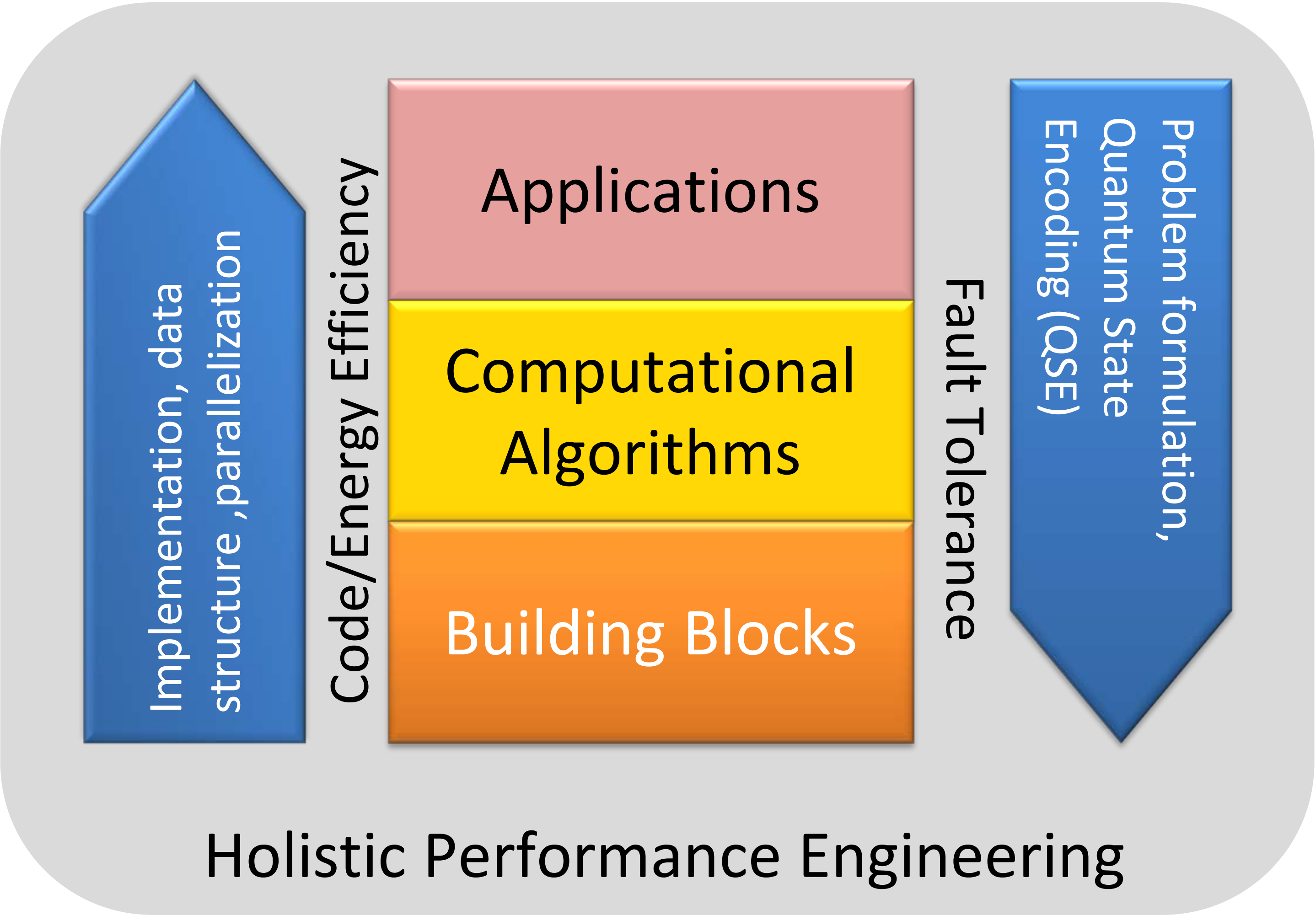
Extremely large sparse matrices: eigenvalues, spectral properties, time evolution

ESSEX

Exascale Sparse Solver Repository (ESSR)

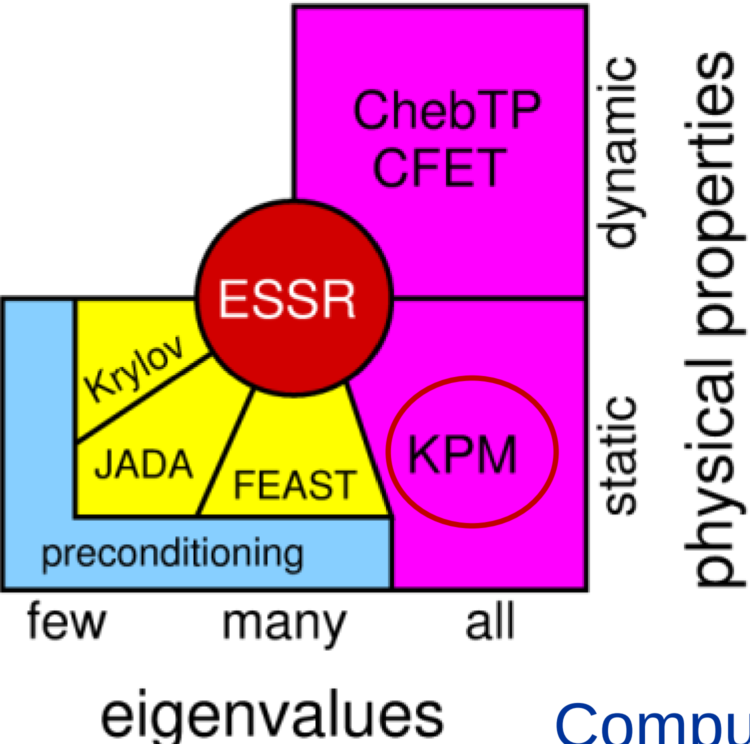
- FT concepts, programming for extreme parallelism
- Sparse eigensolvers, preconditioners, spectral methods
- Quantum physics / chemistry
- ESSEX applications: Graphene, topological insulators, …

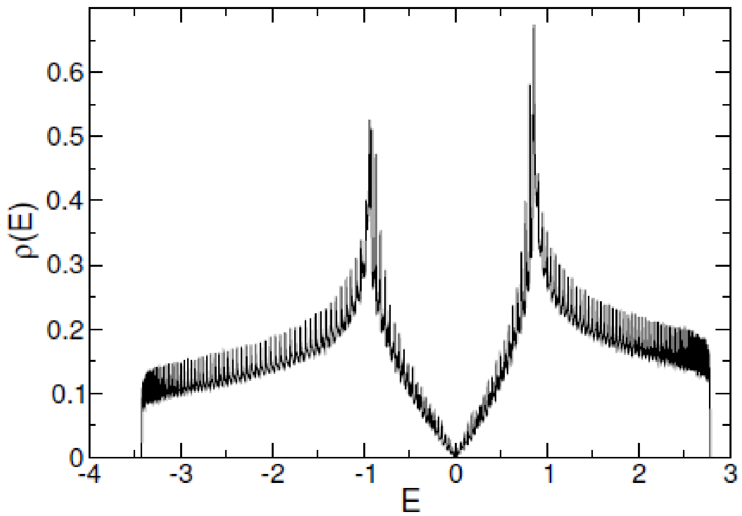
ESSEX: “Co-Design” oriented project



ESSEX: Computational challenges / methods

Cover most aspects of large sparse eigenvalue problem





$$X(\omega) = \frac{1}{N}\mathrm{tr}[\delta(\omega - H)X] = \frac{1}{N}\sum_{n=1}^{N}\delta(\omega - E_n)\langle\psi_n, X\psi_n\rangle$$

Compute approximation to complete eigenvalue spectrum of large sparse matrix A (with $X = I$)

A. Weiße, G. Wellein, A. Alvermann, and H. Fehske, Rev. Mod. Phys. **78**, 275 (2006).
The kernel polynomial method

ESSEX: Start with simple but efficient iterative algorithms ("Kernel Polynomial Method")

for $r = 0$ *to* R-1 **do**
  $|v\rangle = |\text{rand}()\rangle$;
  Initialization &
  computation of μ_0, μ_1
  for $m = 1$ to $M/2$ **do**
    swap($|w\rangle$, $|v\rangle$);
    $|u\rangle = \text{H}|v\rangle$;
    $|u\rangle = |u\rangle - \text{b}|v\rangle$;
    $|w\rangle = -|w\rangle$;
    $|w\rangle = |w\rangle + 2a|u\rangle$;
    $\eta_{2m} = \langle v|v\rangle$;
    $\eta_{2m+1} = \langle w|v\rangle$;
  end
end

Application: R random configurations ($R=1,\ldots,10^2$) or iterative loop

Algorithm: Compute Chebyshev moments

Basic building blocks: spMVM and sparseBLAS1

Checkpoint data: 2 vectors
Constant sparse matrix (H) – recompute

KPM approach can be implemented with only one global communication step

Our (ESSEX) effort –

get a simple prototype solution first –

application driven –

No silver bullet

Fault Tolerance Approaches

1. **Algorithm Based Fault Tolerance (ABFT)**

2. **Message Logging**

3. **Redundancy**

4. **Fault Prediction (*proactive fault tolerance*)**

5. **Checkpoint/Restart (C/R)**

Each of these fault tolerance approaches carries overhead in terms of time and/or resources

J. Hursey. *Coordinated Checkpoint/Restart Process Fault Tolerance for MPI Applications on HPC Systems*. PhD thesis, Indiana University, Bloomington, IN, USA, July 2010.

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

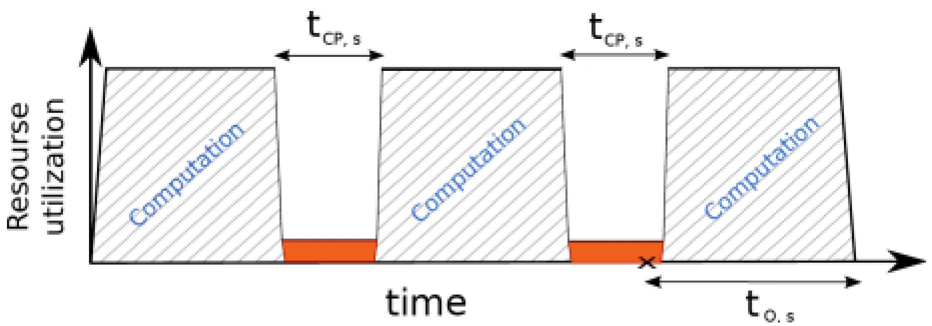
4. Checkpoint compression

5. …

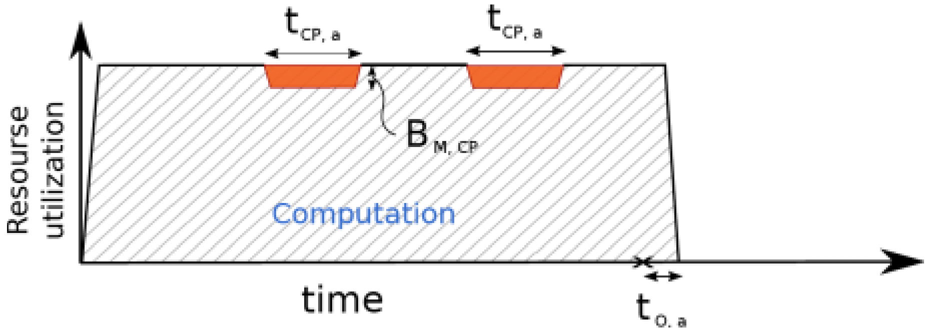
ASYNCHRONOUS CHECKPOINTING

Synchronous vs. asynchronous checkpointing

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

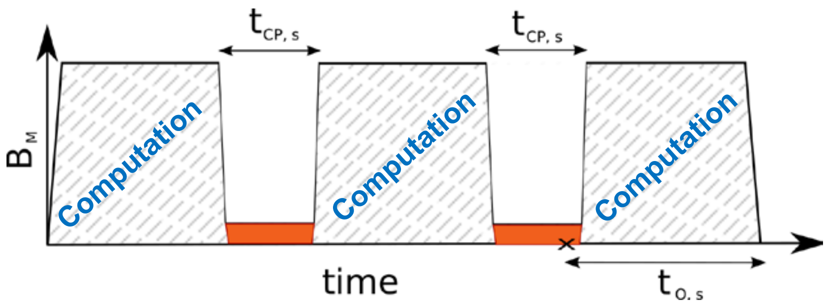


- Asynchronous checkpointing:
  - Dedicated threads for performing asynchronous I/O
  - Low execution time overhead
  - Checkpoint location: flexible (e.g. using SCR)
  - In-memory copy required.



Checkpoint overhead estimation model

Synchronous Checkpointing

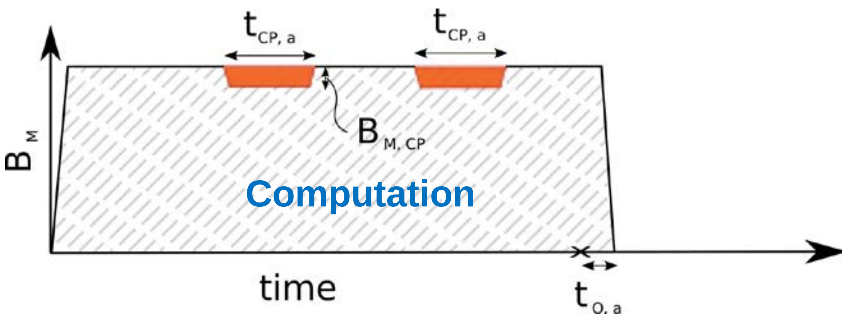


$t_{O,s}$ = overhead for synchronous checkpoints
$t_{CP,s}$ = duration of a synchronous checkpoint
S_{CP} = size of a single checkpoint in bytes
B_{IO} = I/O bandwidth to the file system in bytes/s
B_M = memory bandwidth of a node in bytes/s
n = number of checkpoints

$$t_{O,s} = n \cdot t_{CP,s}$$

$$t_{O,s} = n \cdot \frac{S_{CP}}{B_{IO}}$$

Asynchronous Checkpointing



$t_{O,a}$ = overhead for asynchronous checkpoints
$t_{CP,a}$ = duration of an asynchronous checkpoint
$S_{CP,node}$ = checkpoint size per node in bytes
$B_{M,CP}$ = memory bandwidth used for checkpoint-I/O in bytes/s

$$B_M \cdot t_{O,a} = n \cdot B_{M,CP} \cdot t_{CP,a}$$

$$B_{M,CP} = \frac{m \cdot S_{CP,node}}{t_{CP,a}}$$

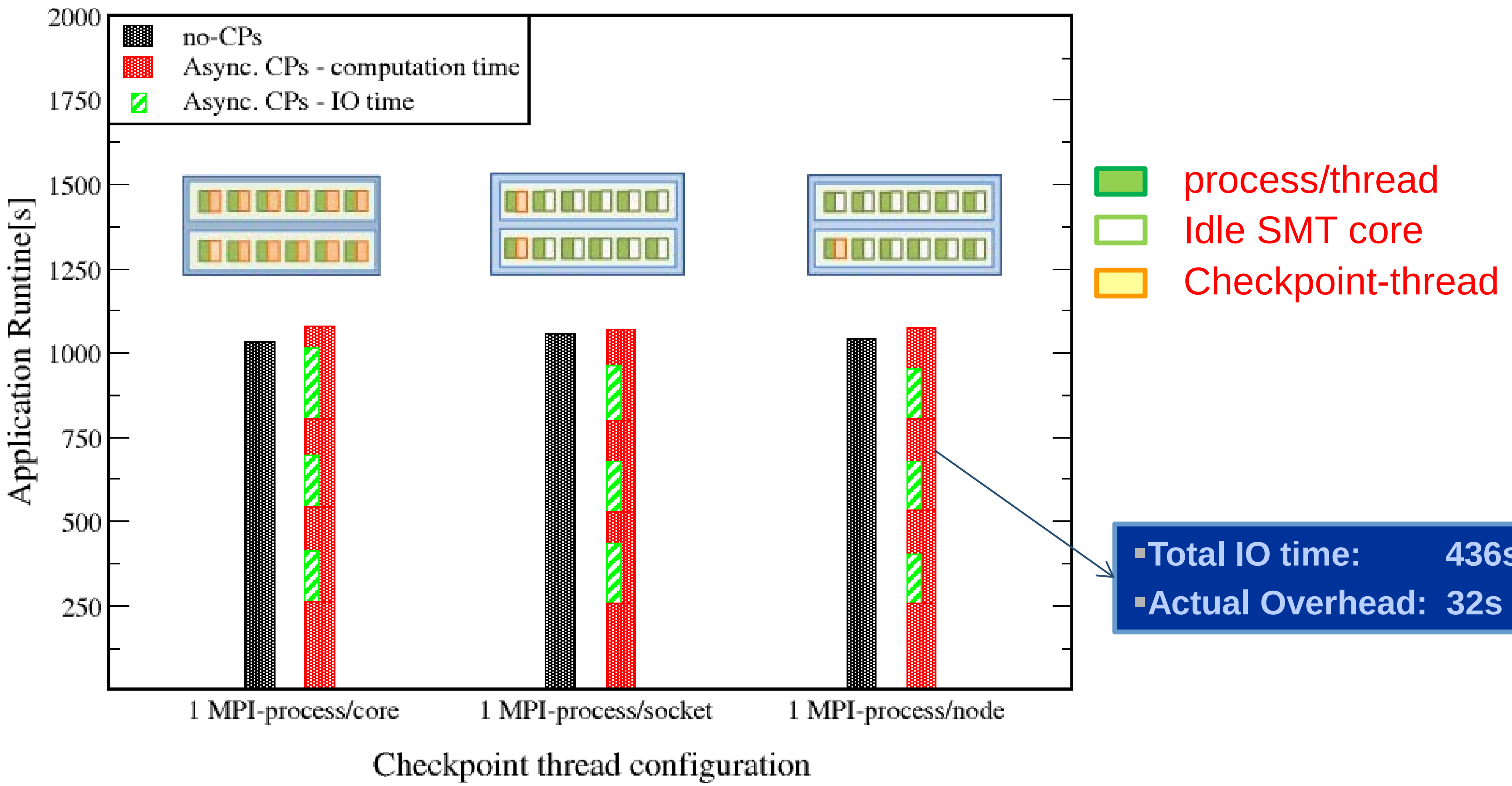
$$t_{O,a} = \frac{m \cdot S_{CP,node}}{B_M} \cdot n$$

F. Shahzad, M. Wittmann, M. Kreutzer, T. Zeiser, G. Hager, and G. Wellein: *A survey of checkpoint/restart techniques on distributed memory systems*. Parallel Processing Letters **23**(04), 1340011-1340030 (2013).

Asynchronous Checkpointing

- Hybrid (MPI-OpenMP) configuration performance comparison

Cluster: LiMa, num. of nodes = 32, PFS = LXFS, Aggregated CP size = 200 GB/CP



- Total IO time: 436s
- Actual Overhead: 32s

Basic building blocks library: GHOST

General, Hybrid and Optimized Sparse Toolkit

GHOST

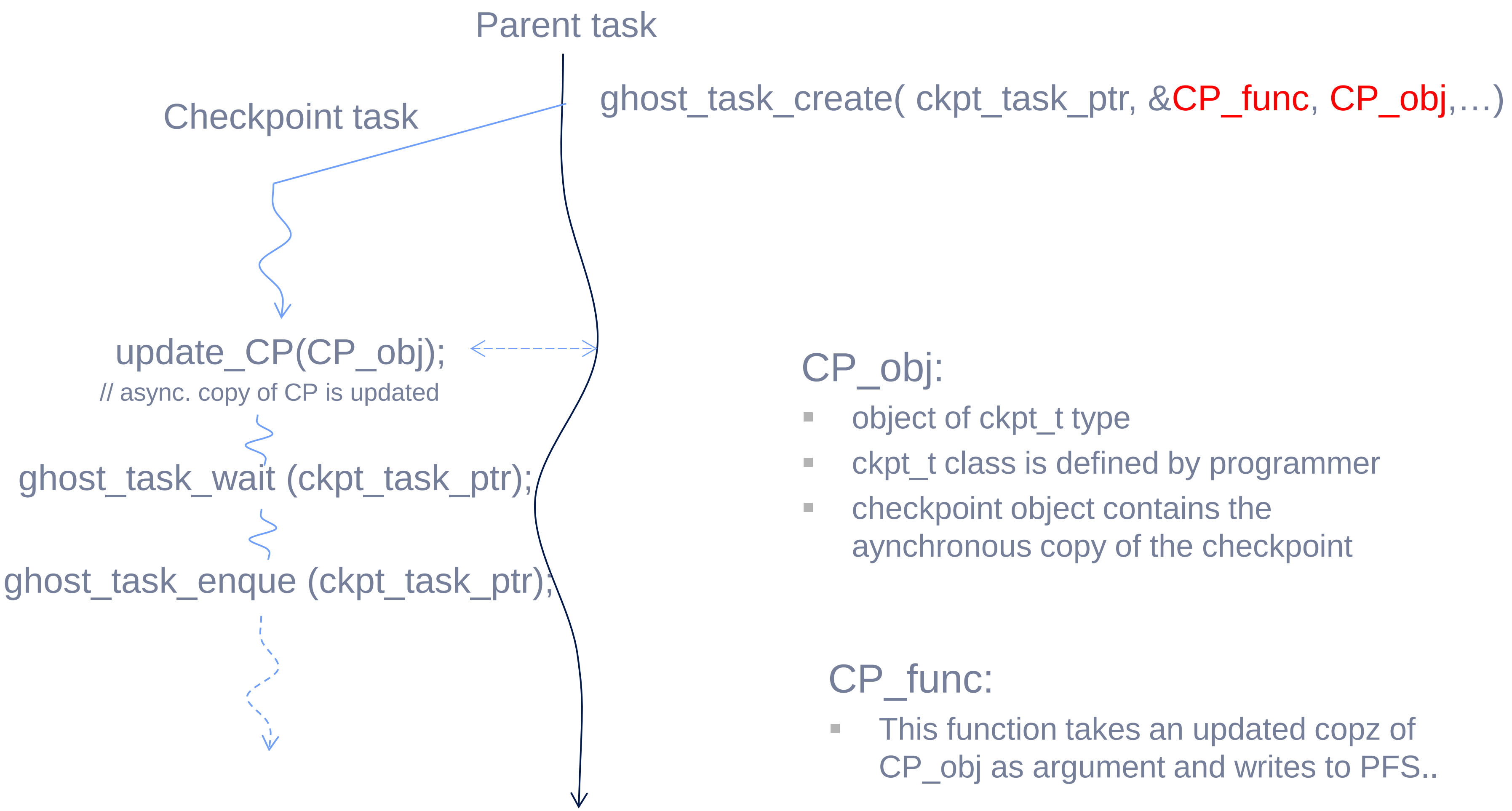
- Basic tailored sparse matrix / vector operations
- CRS or **SELL-C-σ*** (**unified format**) storage schemes
- (Block-)SpMVM: SIMD intrinsic (AVX, SSE, MIC) & CUDA kernels
- Dense vector /matrices: row-/column-major storage

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

APPLICATION DRIVEN AUTOMATIC FAULT TOLERANCE (AFT)

Our (naïve) approach:

- Regular asynchronous Checkpoints (FS or remote node)
- Node failure detected by communication library
  (Communication library in valid state after node/process loss)
- Spare nodes are available – application replaces lost node
- Application driven restart from last checkpoint

FT communication libraries

A long time ago it was no problem to

- tolerate the frequent loss of processes/nodes
- register new processes/nodes dynamically on demand

→**P**arallel **V**irtual **M**achine (PVM)

Today:

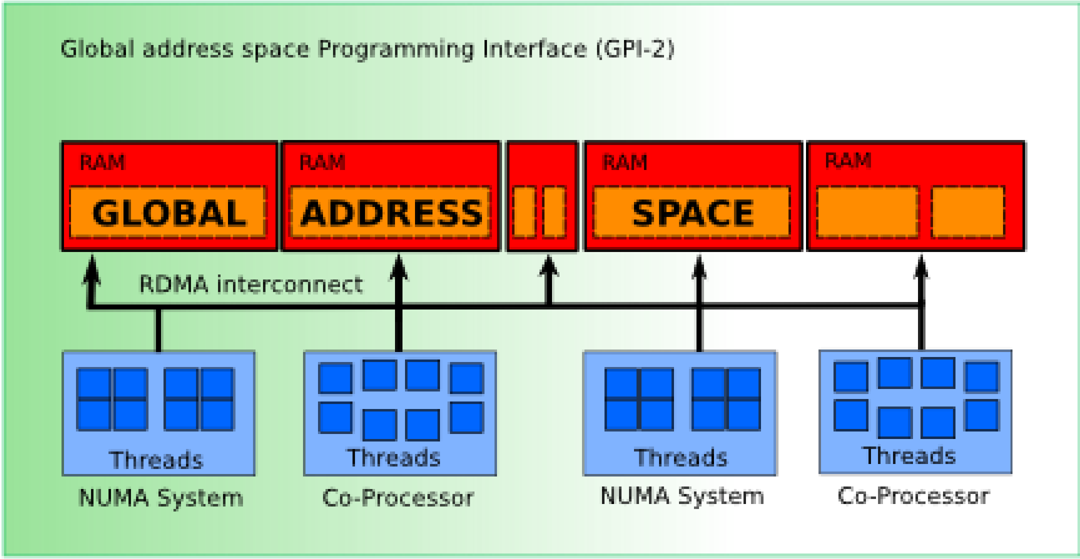
- Several non-standard libraries, e.g. Charm++ or GPI

- Why is FT not part of MPI?
  - Complexity of MPI standard / MPI forum?
  - Restrict FT feature on small parts of MPI standard?

AFT: GPI Introduction

- Current version: **GPI-2** (see http://www.gpi-site.com/gpi2/)
  Developed by Fraunhofer IWTM
- Implements **GASPI** standard: http://www.gaspi.de/software.html
  (Global Address Space Programming Interface)
- PGAS programming model

- Two memory parts
  - Local memory:
    local to each GPI process
  - Global memory: Accessible
    for other processes



- Enables fault tolerance
  - via providing TIMEOUT for every communication call.

AFT: GPI - Application requirements

- **Algorithm based on PGAS model**

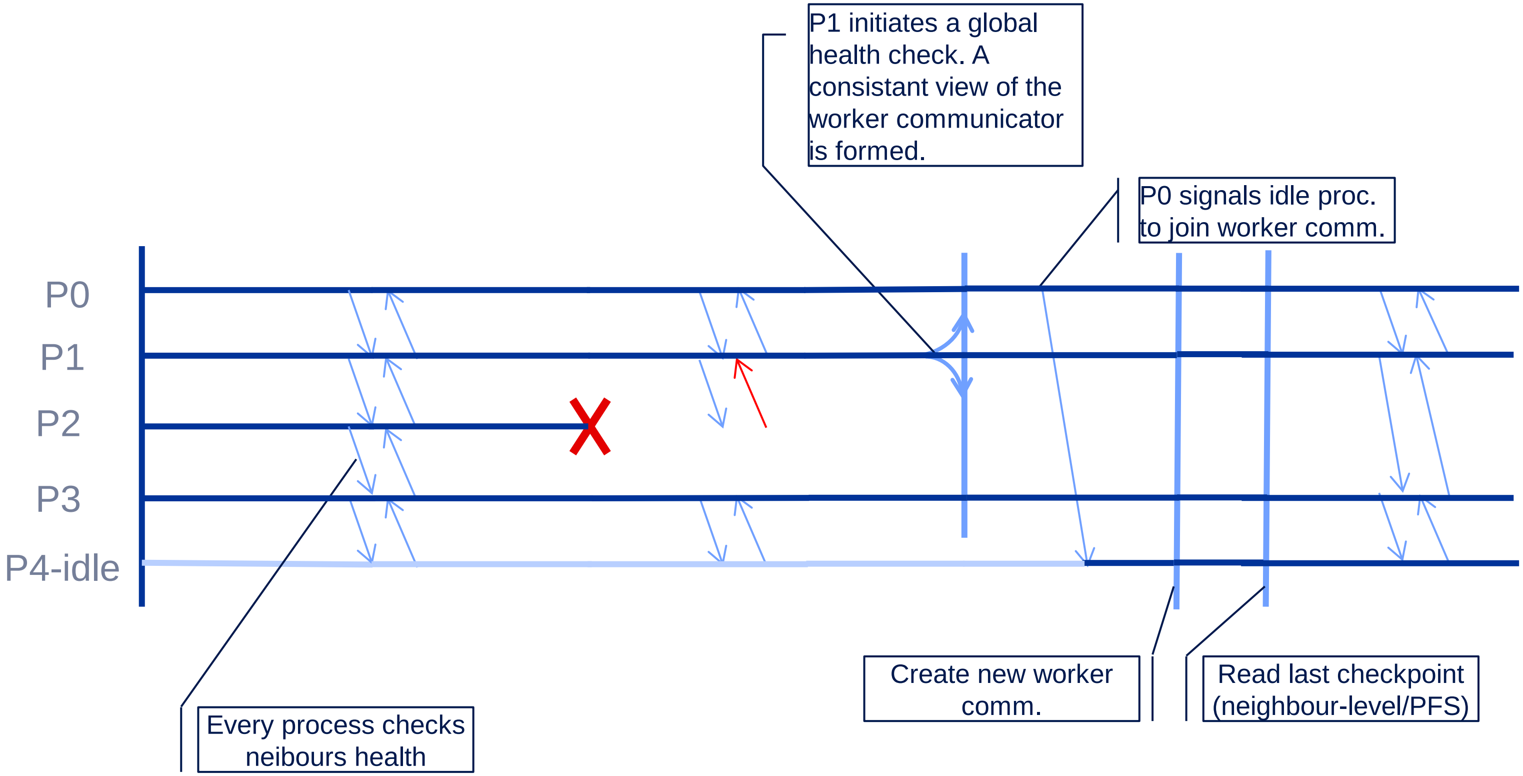
- **For effective fault tolerance**
  - No global synchronization, barriers
  - Each GPI-process communicates with certain subset of GPI-processes (e.g. neighbors)
  - In case of failures, rest of the processes detect errors in results and correct them accordingly.

- **Algorithm driven FT based applications**

Prototype FT implementation

- Idea:
  - Running the program with ‚n+m' processes, where ‚m' is the number of idle processes.
  - Program initially utilizes ‚n' processes for work (work-group)
  - In case of a failed process in ‚work-group', an idle process is added to the ‚work-group'.
  - Processes in newly established ‚work-group' restart the work from last checkpoint.

GPI FT program flow:



GPI fault recovery overhead :

- Timeout returns for communication after failure
  - Only the communication to/from the failed process contibutes to this overhead.
- Global health vector update to have consistant view of the health vector across all processes
- Rebuilding worker communicator
  - Process 0 signals the idle processes, which then joins the creation of new comm.
- Checkpoint fetching from neighbour (or PFS) and reinitializing

Testing:

- Tested successfully up to 1000+ cores with 1-2 failures.

- Challenges using higher number of cores:
  - Seg. fault during deletion of old comm/ recreat new comm.
  - Issue using barrier for new comm.
  - Both issues are under investigation by Frauenhofer IWTM.

→ Bug in GPI library has been detected and will be fixed in next release.

Concluding remarks:

If you use checkpointing

- do it asynchronously
- use dedicated threads
- use application specific knowledge

- restarting at runtime is a challenge with current communication libraries → You feel like a test pilot

GPI is on a reasonable way

Exascale “modus operandi” still unclear:

- Pool of spare nodes?
- Continue wit remaining set

GHOST will become public available this year

–

If you are interested in testing, you are very welcome

–

ask us: https://blogs.fau.de/essex/

Thank you!

Questions?

SPPEXA

Partially funded by DFG Priority Programme1648

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

Partially funded by BMBF project FeTol

The seminar topic

How many users (scientific communities) need within a decade?

Which of them need?

Resilience in
Exascale Computing

Capacity?
10.000 jobs@10 nodes

Capability!
1 job@100,000 nodes

Where is the sweet spot?

The seminar topic

Do we need resilience beyond 2025?

Who ensures/guarantees?

Hardware?

Resilience in Exascale Computing

FT algorithms?

Does overhead pay off?

Low level automatic SW solution – silver bullet?

Application with OS/HW support?

Conservation law of HPC
Flexibility * Performance = constant