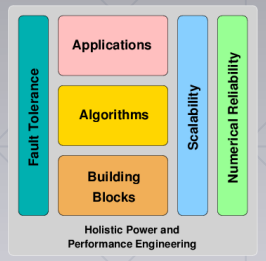
ESSEX II –
Equipping Sparse Solvers for Exascale

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



SPPEXA

Holger Fehske
Institute for Physics, U. Greifswald

Bruno Lang
Applied Computer Science, U. Wuppertal

Georg Hager
Erlangen Regional Computing Center

Achim Basermann
Simulation & SW Technology, DLR

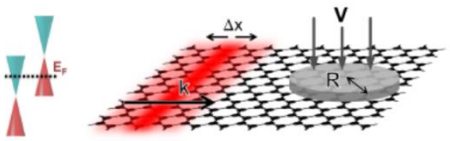
Gerhard Wellein
Computer Science, U. Erlangen

Tetsuya Sakurai
Applied Math., U. Tsukuba

Kengo Nakajima
Computer Science, U. Tokyo

ESSEX project – background

Quantum physics/information applications



Large, Sparse

$$i\hbar\frac{\partial}{\partial t}\Psi = H\Psi \qquad i\hbar\frac{\partial}{\partial t}\varrho = [H,\varrho] + D$$

and beyond….

$$\boldsymbol{H\,x = \lambda\,x}$$

"Few" (1,…,100s) of eigenpairs

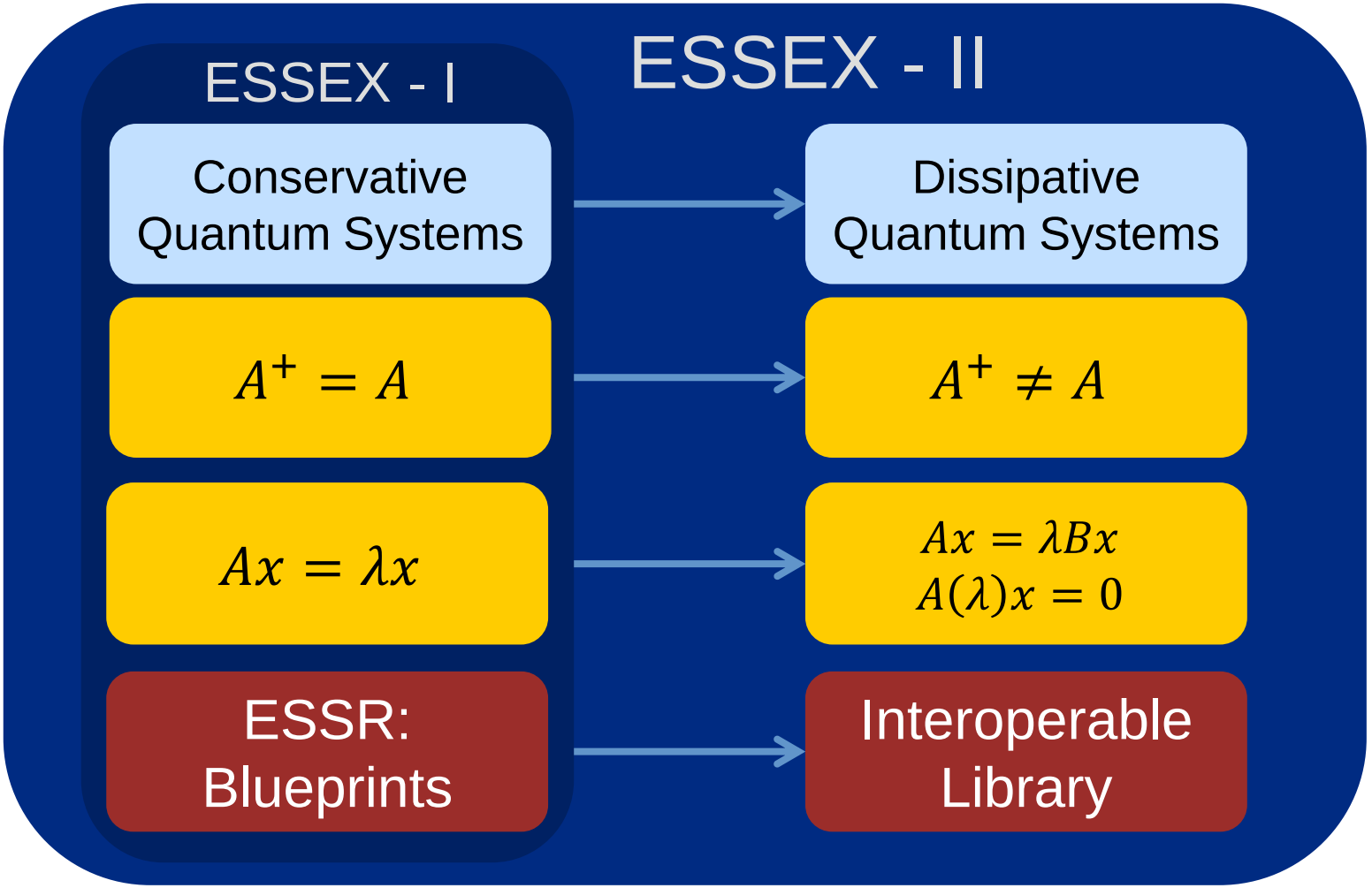
"Bulk" (100s,…,1000s) eigenpairs

$$\{\lambda_1, \lambda_2, \dots, \dots, \dots, \dots, \lambda_k, \dots, \dots, \dots, \dots, \lambda_{n-1}, \lambda_n\}$$

Good approximation to full spectrum (e.g. Density of States)

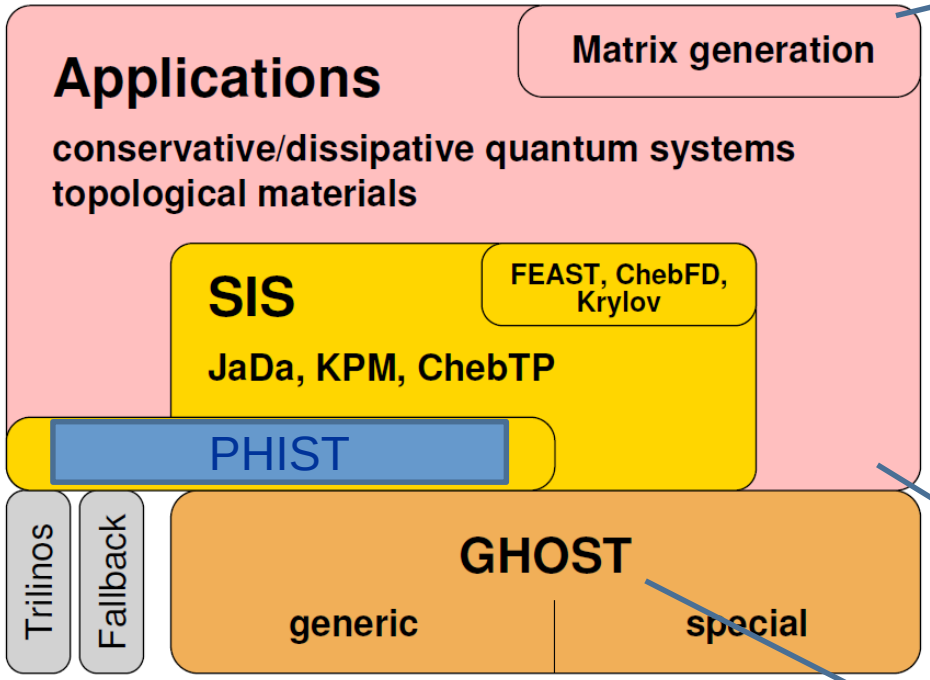
→ Sparse eigenvalue solvers of broad applicability

Motivated by quantum physics applications



→ Sparse eigenvalue solvers of broad applicability

ESSEX-II: Focus topics in 2016/17



Scalable matrix generation for a collection of quantum physics problems

Impact of enhanced / reduced accuracy

Implementation & evaluation of various preconditioning strategies

CRAFT library:
Application-level Checkpoint/Restart & Automatic Fault Tolerance (MPI-ULFM)

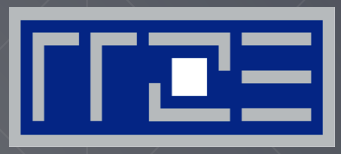
TSMM@GPU kernels

P100/KNL porting / optimization

GHOST: General Hybrid Optimized Sparse Toolkit

PHIST: Pipelined Hybrid Parallel Iterative Solver Toolkit

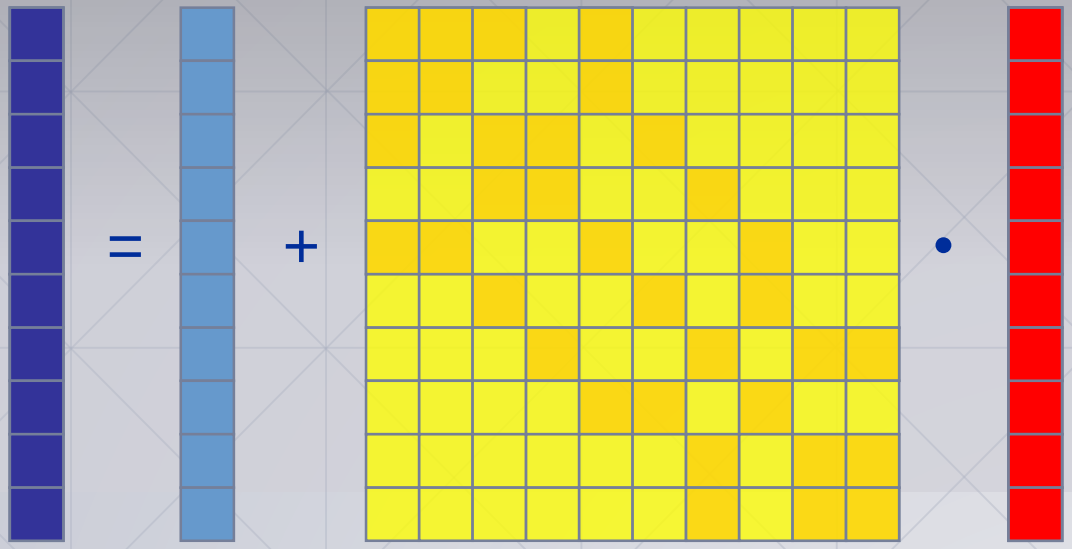
ERLANGEN REGIONAL
COMPUTING CENTER



Solvers & Hardware efficiency

SPARSE MATRIX DATA FORMAT – ONE SIZE FITS ALL

Basic kernel Sparse Matrix Vector Multiply (SpMVM)



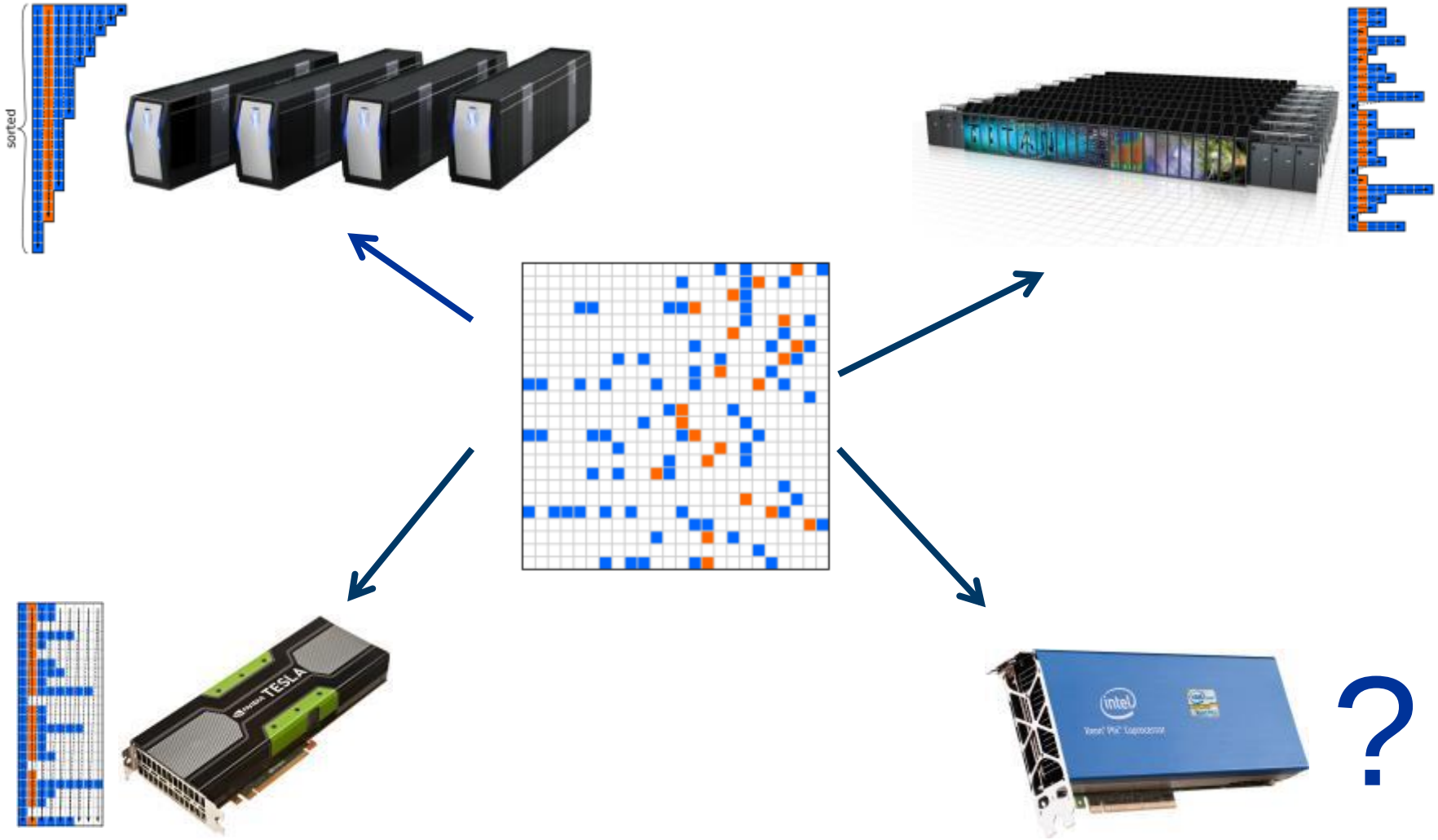
Which matrix data format?

Is there a single one?

48 page review* on formats for GPUs only

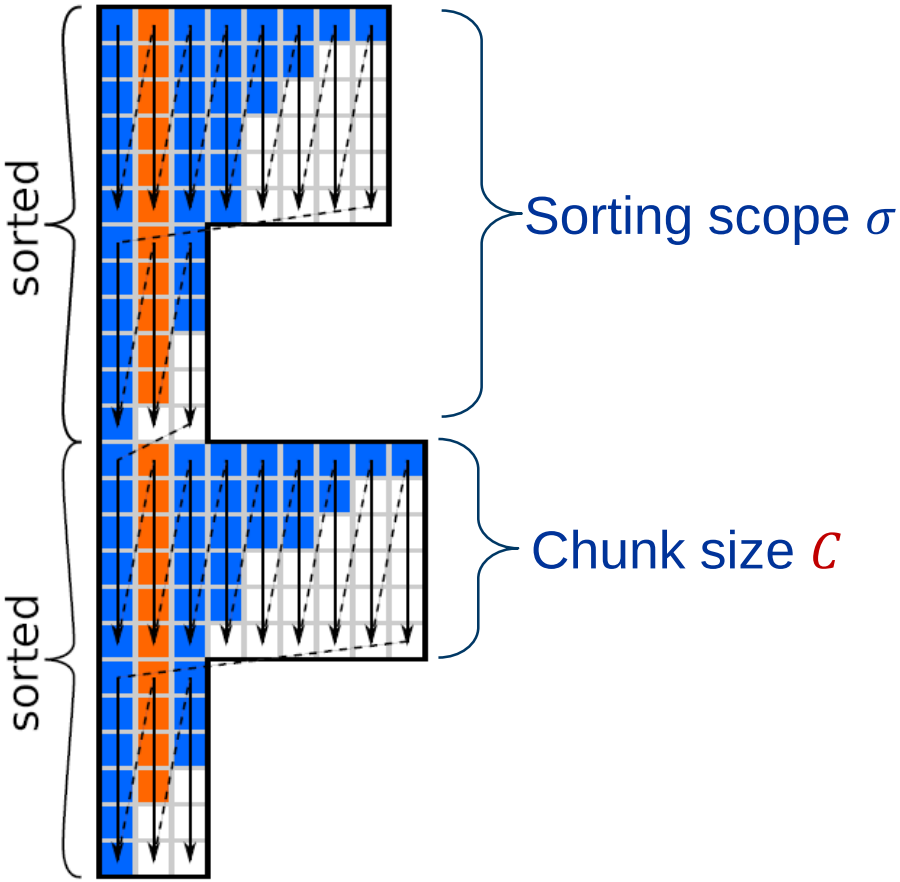
*Salvatore Filippone, Valeria Cardellini, Davide Barbieri, and Alessandro Fanfarillo. 2017. Sparse Matrix-Vector Multiplication on GPGPUs. *ACM Trans. Math. Softw.* 43, 4, Article 30 (January 2017). DOI: https://doi.org/10.1145/3017994

Sparse matrix formats:
Which one to choose for some architecture?



SELL-C-σ: Combining them all into one

1. Pick chunk size C (guided by **SIMD/SIMT/vector** widths)
2. Pick **local sorting** scope σ
3. Sort rows by length within each sorting scope (local JDS)
4. Pad chunks with zeros to make them rectangular (Sliced ELLPACK)
5. Store matrix data in “chunk column major order” (CRS-type)

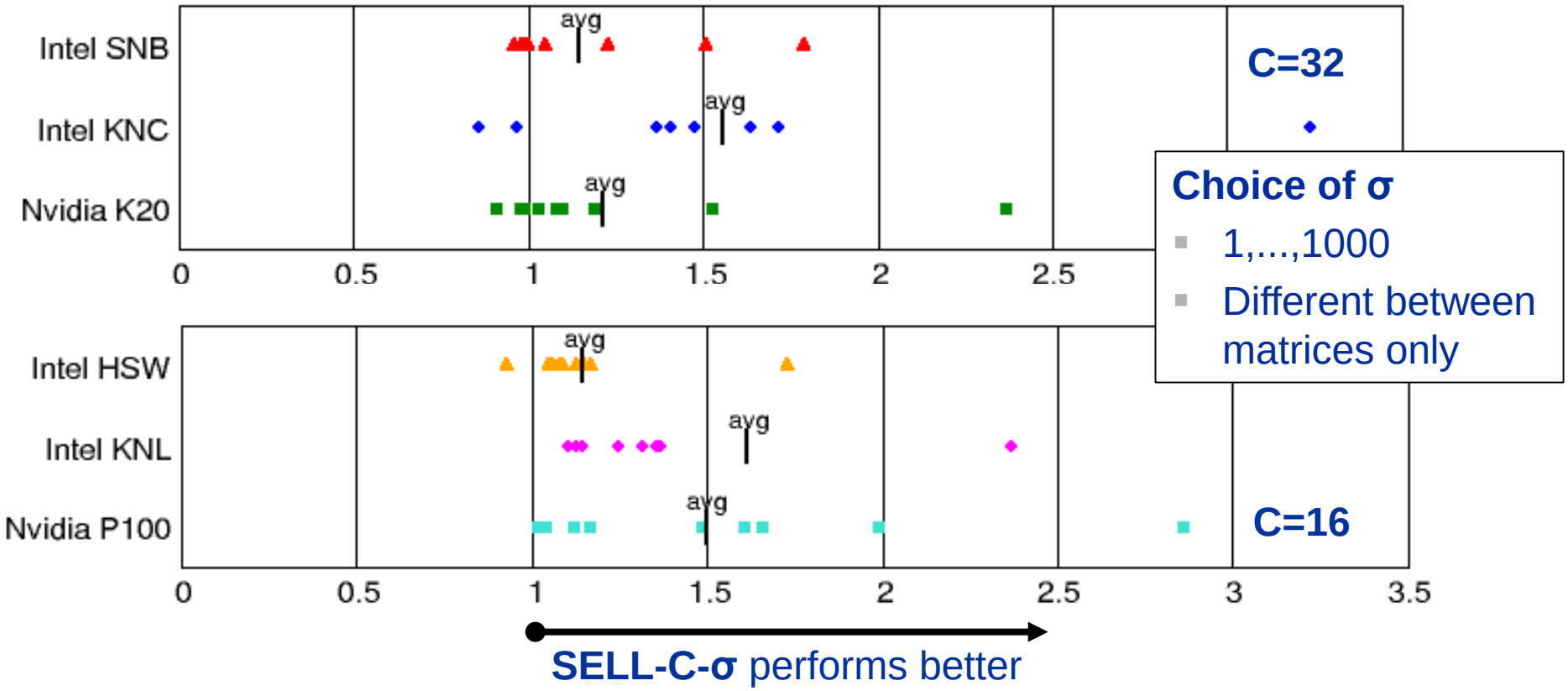


M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop: **A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units**. SIAM Journal on Scientific Computing 36(5), C401–C423 (2014). DOI: 10.1137/130930352

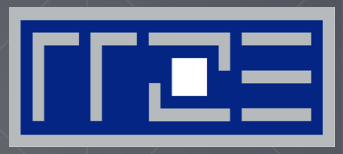
SELL-C-σ: One size fits all – and over time

SELL-C-σ performance vs. baselines (10 corner cases UoF collection):

- CRS for SNB and Xeon Phi (using latest MKL)
- HYB for Nvidia (using latest CUSPARSE)

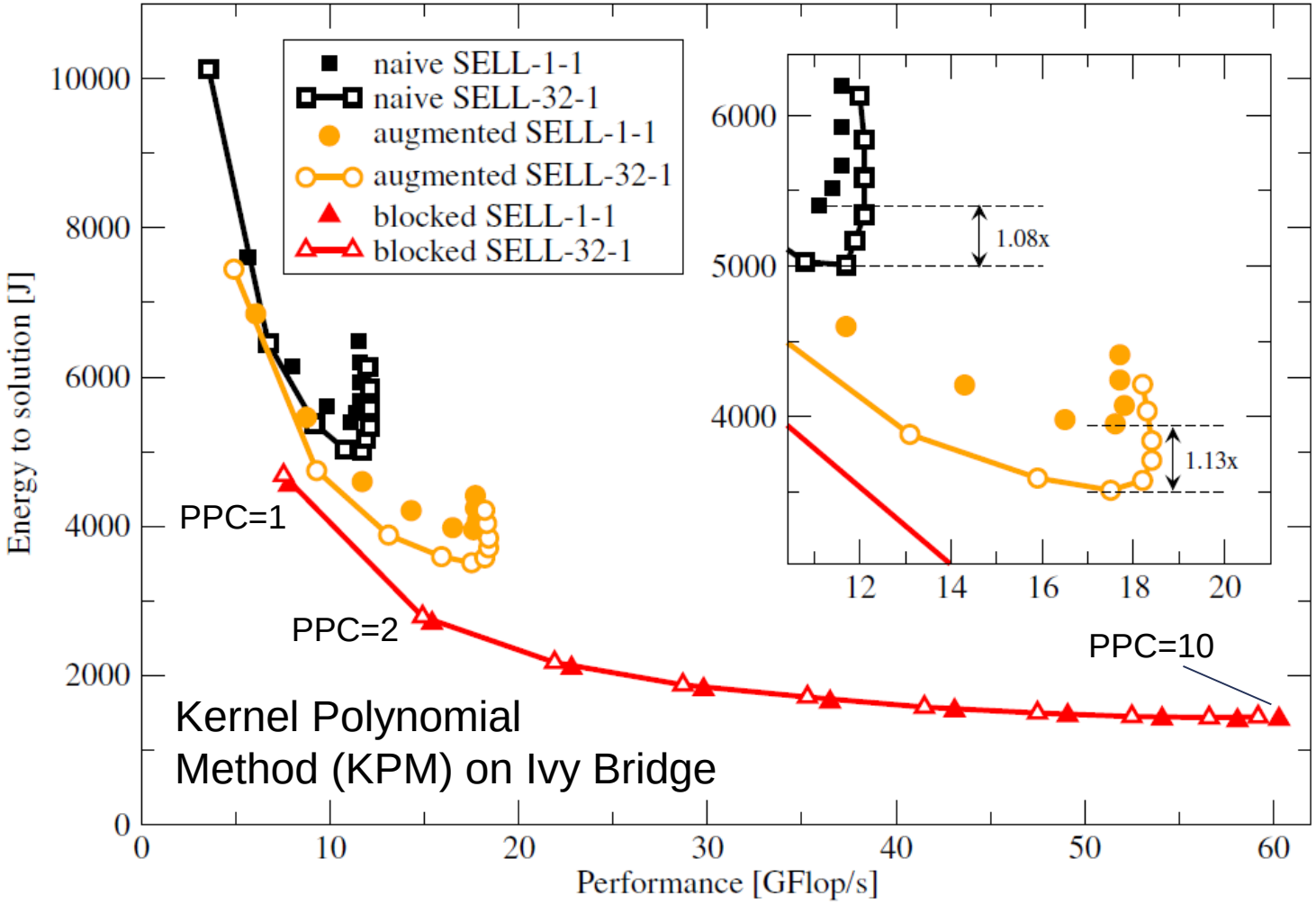


ERLANGEN REGIONAL COMPUTING CENTER



Hardware efficient solvers & applications

Energy impact of data format and optimizations: The “Z-plot”



Hardware efficient (linear) solvers
Kaczmarz solver (used in BEAST-C)

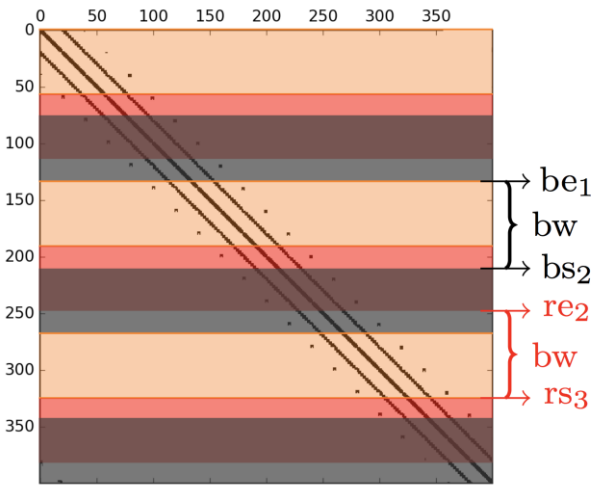
```
for row = 0 : nrows do
  scale = b[row]
  norm = 0
  for idx = rowptr[row] : rowptr[row + 1] do
    scale− = val[idx] ∗ x[col[idx]]
    norm+ = val[idx] ∗ val[idx]
  end for
  scale∗ = omega/norm
  for idx = rowptr[row] : rowptr[row + 1] do
    x[col[idx]]+ = scale ∗ val[idx]
  end for
end for
```

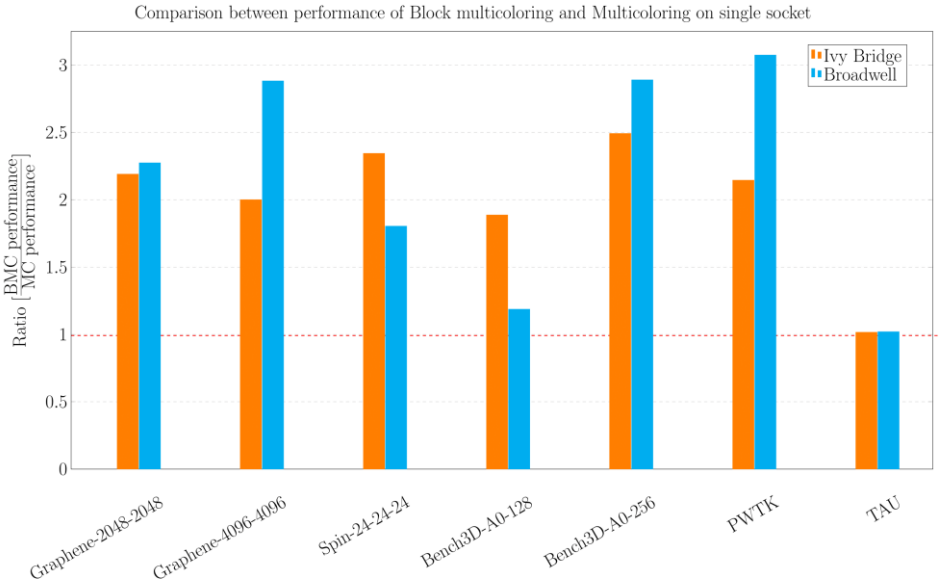
$$x^{k+1} = x^k + \omega * \frac{(b_i - < A_i, x^k >)}{\|A_i\|^2} * A_i^T$$

Parallelization

Standard: D2-multicoloring (distance: 2) (D2-MC)

New: block-multicoloring (BMC)
(+ D2-MC, where needed)

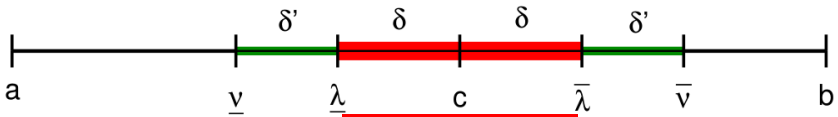




C.L. Alappat, Implementation and Performance Engineering of the Kaczmarz Method for Parallel Systems, Master Thesis

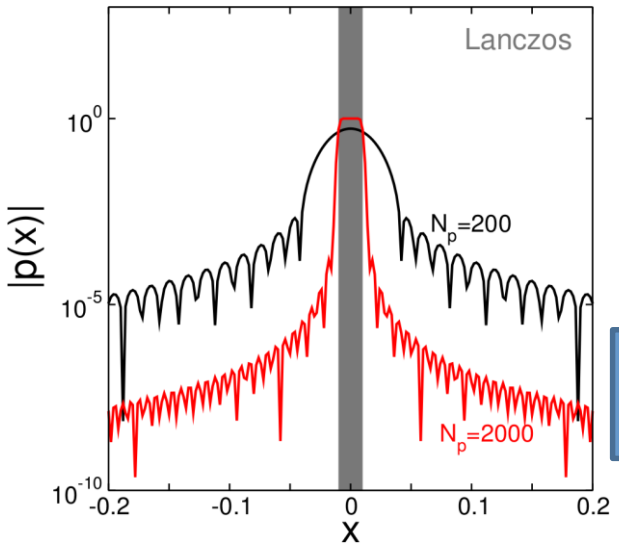
Chebyshev filter diagonalization: Algorithm & Performance (I)

Many interior eigenvalues of large sparse (Hermitian) matrices

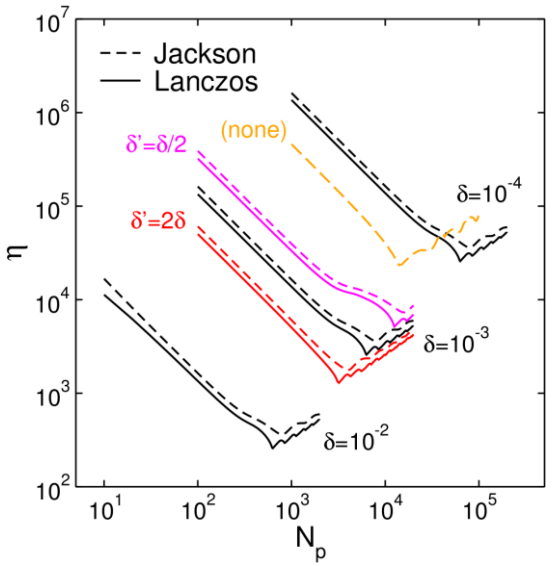


Target

Search Space



Filter quality (lower is better)



uses only spMVM / low synch. ⇒ very large unstructured matrices

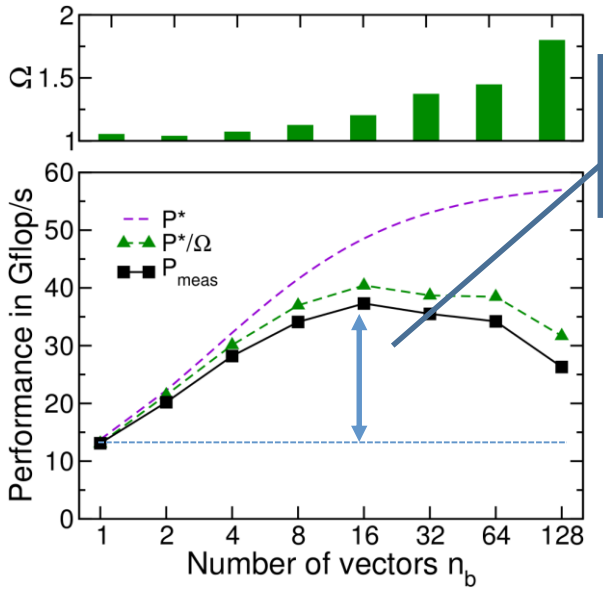
convergence ←→ eigenvalue density ⇒ large number of spMVMs

clue to convergence: search vectors ≫ target vectors

⇒ massive vector blocking ⇒ spMMVM

Chebyshev filter diagonalization: Algorithm & Performance (II)

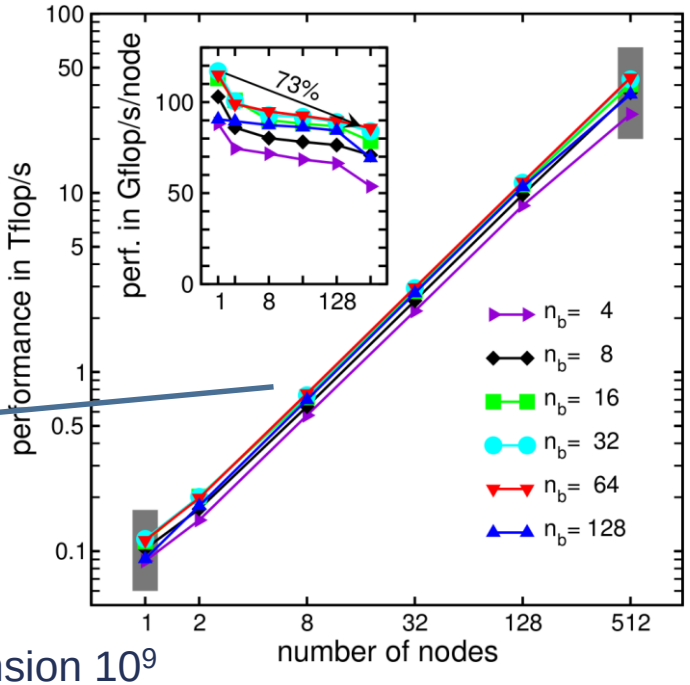
HSW-socket performance



Vector blocking
3x performance gain

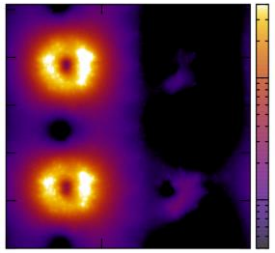
Weak scaling for topological insulator testcase

SuperMUC-2 performance



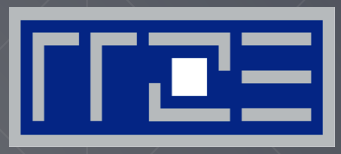
Compute 200 interior eigenpairs of a matrix of dimension 10^9 (10 hrs on 512 nodes)

Matrix	Nodes	D	$[\underline{\lambda} : \overline{\lambda}]_{\text{rel}}$	N_T	N_p	Runtime [hours]	Sust. perf. [Tflop/s]
topi1	32	6.71e7	7.27e−3	148	2159	3.2 (83%)	2.96
topi2	128	1.07e9	3.64e−3	148	4319	4.9 (88%)	11.5
topi3	512		1.82e−3	148	8639	10.1 (90%)	43.9
graphene1	128		4.84e−4	104	32463	10.8 (98%)	4.6
graphene2	512	1.07e9	2.42e−4	104	64926	16.4 (99%)	18.2



Pieper, Kreutzer, Alvermann, Galgon, Fehske, Hager, Lang, Wellein, J. Comp. Phys. 325, 226 (2016)

ERLANGEN REGIONAL COMPUTING CENTER



CRAFT: A library for application-level Checkpoint/Restart Automatic Fault Tolerance

F. Shahzad, J. Thies, M. Kreutzer, T. Zeiser, G. Hager, and G. Wellein: *CRAFT: A library for easier application-level checkpoint/restart and automatic fault tolerance.* Submitted. Preprint: arXiv:1708.02030

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

CRAFT Introduction:

1. Checkpoint/restart(CR):
   - Target: A simple and extendable library tool for creating “Application-level checkpoints” with minimal code modifications
   - Default checkpointable data types: i) PODs(int, double etc.) ii) POD arrays iii) POD multiarrays iv) MPI derived data types
   - Extendable to include more datatypes
   - Signal initiated checkpointing (SIGALRM)
   - Support for multi-level and nested checkpoints
   - Takes care of checkpoint management issues (e.g. restart checkpoint version)
   - Optimizations:
     i. Asynchronous checkpointing.
     ii. MPI-IO for PFS-level checkpoints.
     iii. Supports SCR-library (LLNL) for neighbor-level C/R.
2. Automatic fault tolerance (AFT):
   - Dynamic process failure recovery in case of processor failures*

* Terms and conditions apply

CR: Toy example

A toy code without (left) and with CRAFT application-level CR functions (right).

```
#include <mpi.h>

int main(int argc, char* argv[]){
  int n=5, iteration=1;
  double dbl = 0.0;
  int * dataArr    = new int[n];









  for(; iteration <= 100 ; iteration++){
    // Computation-communication loop
    modifyData(&dbl, dataArr);

  }
  return EXIT_SUCCESS;
}
```

```
#include <mpi.h>
#include <craft.h>
int main(int argc, char* argv[]){
  int n=5, iteration=1, cpFreq=10;
  double dbl = 0.0;
  int * dataArr    = new int[n];
  // ===== DEFINE CHECKPOINT ===== //
  Checkpoint myCP("myCP", MPI_COMM_WORLD);
  myCP.add("dbl", &dbl);
  myCP.add("iteration", &iteration);
  myCP.add("dataArr", dataArr, n);
  myCP.commit();
  myCP.restartIfNeeded(&iteration);
  for(; iteration <= 100 ; iteration++){
    // Computation-communication loop
    modifyData(&dbl, dataArr);
    myCP.updateAndWrite(iteration, cpFreq);
  }
  return EXIT_SUCCESS;
}
```

Checkpoint: A collection of data objects of checkpointable types.

Checkpointable data-type: A data-type that is recognized by add() function of CRAFT.

Automatic fault tolerant (AFT):

- AFT: Dynamic process(es) recovery in case of process failure(s)

1. A fault tolerant communication (avoids deadlocks in case of failed processes)
   - ULFM-MPI
   - Error handler / MPI call return value
   - Error propagation via MPI_Comm_revoke()

2. Communicator recovery
   - Shrinking/Spawning

3. Data recovery
   - Easy option: Checkpoint/Restart
   - Algorithm Based Fault Tolerance

Automatic fault tolerance (AFT):

- An “AFT zone” is created between AFT_BEGIN() and AFT_END()
- The process failures (of the given communicator) within the AFT-zone are recovered dynamically.
- Communicator Recovery options:
  1. Shrinking
  2. Non-shrinking

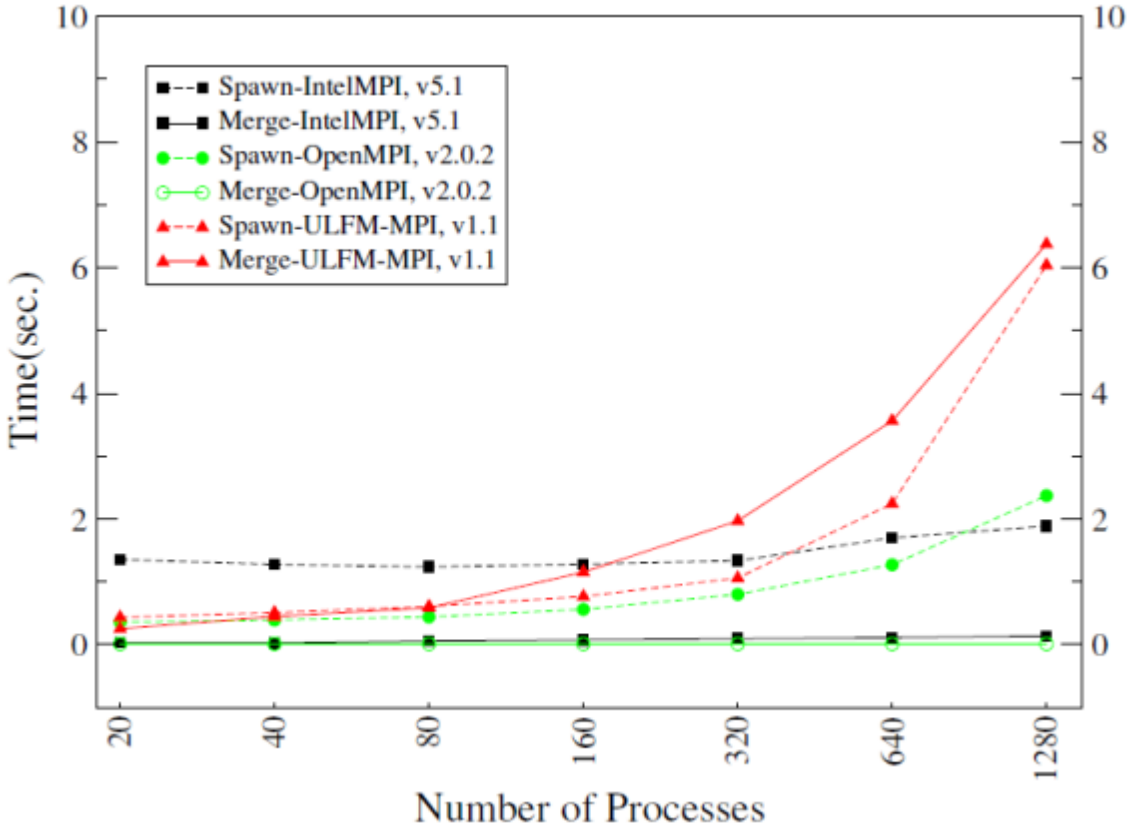
```
#include <craft.h>
int main(int argc, char* argv[])
{
  ...
  int myrank;
  MPI_Comm FT_Comm;
  MPI_Comm_dup(MPI_COMM_WORLD, &FT_Comm);
  AFT_BEGIN(FT_Comm, &myrank, argv);
  double data = 0;
  int iteration = 0, cpFreq = 10;
  Checkpoint myCP( "myCP", FT_Comm);
  myCP.add("data", &data);
  myCP.add("iteration", &iteration);
  myCP.commit();

  if( myCP.needRestart() ) {myCP.read();}
  for(; iteration <= n; iteration++)
  {
    // Computation-communication loop
    if(iteration % cpFreq == 0)
      { myCP.update();  myCP.write();}
  }
  ...
  AFT_END();
}
```

AFT zone

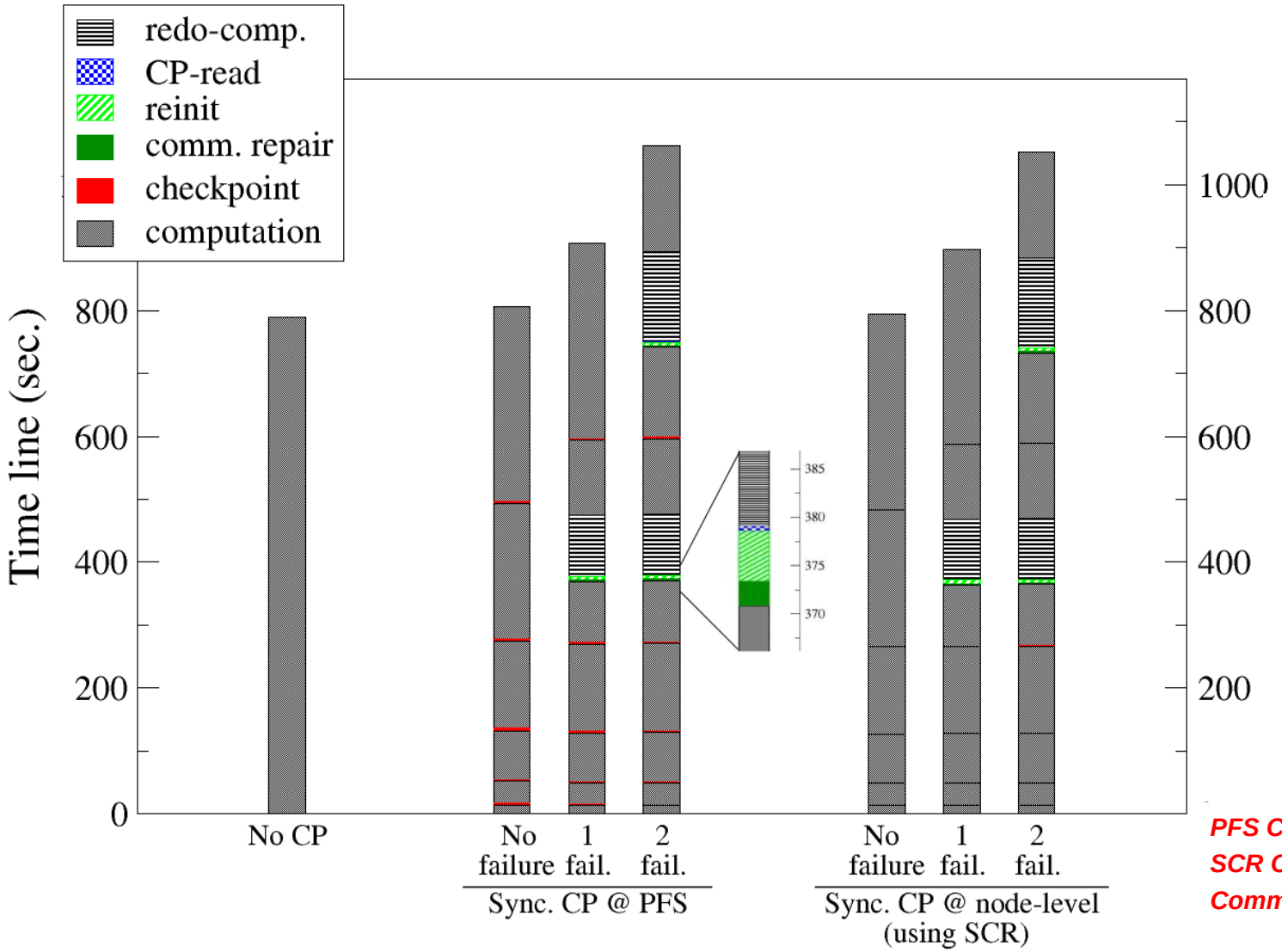
CRAFT Benchmarks: spawn + merge comparison

- A scaling comparison of spawn and merge routines for Intelmpi-v5.1 vs. OMPI-v2.0.2 vs. ULFM-1.1 implementations.



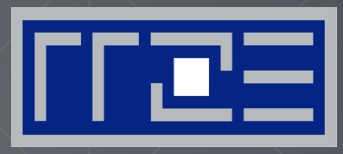
- Better Spawn-Merge results in future with porting of ULFM into new Open MPI branches.

CRAFT Benchmarks: Lanczos, 128 Emmy nodes(-np 256)



- *Num. Nodes = 128*
- *np = 256 (npernode=2)*
- *10 threads/process*
- *Recovery=Non-shrinking*

PFS CP Overhead= ~1.26%
SCR CP Overhead= ~0.56%
Comm. Rec. overhead=~2.6sec.



Thank You.

DFG Priority Programme1648
ESSEX-II
(see www.sppexa.de)

Links

- GHOST / PHIST / CRAFT (ESSEX software)
  https://bitbucket.org/essex
- Equipping sparse scalable solvers for exascale (ESSEX):
  https://blogs.fau.de/essex/

References - ESSEX

- A. Pieper, M. Kreutzer, A. Alvermann, M. Galgon, H. Fehske, G. Hager, B. Lang, and G. Wellein: High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. Journal of Computational Physics 325, 226-243 (2016). DOI: 10.1016/j.jcp.2016.08.027, Preprint: arXiv:1510.04895

- M. Röhrig-Zöllner, J. Thies, M. Kreutzer, A. Alvermann, A. Pieper, A. Basermann, G. Hager, G. Wellein, and H. Fehske: Increasing the performance of the Jacobi-Davidson method by blocking. SIAM Journal on Scientific Computing, 37(6), C697–C722 (2015). DOI: 10.1137/140976017, Preprint: http://elib.dlr.de/89980/

- M. Kreutzer, G. Hager, G. Wellein, A. Pieper, A. Alvermann, and H. Fehske: Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems. Proc. IPDPS15. DOI: 10.1109/IPDPS.2015.76, Preprint: arXiv:1410.5242

- M. Kreutzer, G. Hager, G. Wellein, H. Fehske, and A. R. Bishop: A unified sparse matrix data format for modern processors with wide SIMD units. SIAM Journal on Scientific Computing 36(5), C401–C423 (2014). DOI: 10.1137/130930352, Preprint: arXiv:1307.6209

- M. Kreutzer, J. Thies, A. Pieper, A. Alvermann, M. Galgon, M. Röhrig-Zöllner, F. Shahzad, A. Basermann, A. R. Bishop, H. Fehske, G. Hager, B. Lang, G. Wellein: Performance Engineering and Energy Efficiency of Building Blocks for Large, Sparse Eigenvalue Computations on Heterogeneous Supercomputers. Springer LNCS Volume 113 (2016). DOI: 10.1007/978-3-319-40528-5_14

- M. Galgon, L. Krämer, B. Lang, A. Alvermann, H. Fehske, A. Pieper, G. Hager, M. Kreutzer, F. Shahzad, G. Wellein, A. Basermann, M. Röhrig-Zöllner, and J. Thies: Improved Coefficients for Polynomial Filtering in ESSEX. Accepetd for publication in Proc. EPASA 2015