ESSEX: Equipping Sparse Solvers for Exascale

Andreas Alvermann[1], Achim Basermann[2], Holger Fehske[1], Martin Galgon[3], Georg Hager[4], Moritz Kreutzer[4], Lukas Krämer[3], Bruno Lang[3], Andreas Pieper[1], Melven Röhrig-Zöllner[2], Faisal Shahzad[4], Jonas Thies[2], and Gerhard Wellein[4]

[1] Ernst-Moritz-Arndt-Universität Greifswald, Greifswald, Germany
[2] German Aerospace Center, Köln, Germany
[3] Bergische Universität Wuppertal, Wuppertal, Germany
[4] Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany

Abstract. The ESSEX project investigates computational issues arising at exascale for large-scale sparse eigenvalue problems and develops programming concepts and numerical methods for their solution. The project pursues a coherent co-design of all software layers where a holistic performance engineering process guides code development across the classic boundaries of application, numerical method, and basic kernel library. Within ESSEX the numerical methods cover widely applicable solvers such as classic Krylov, Jacobi-Davidson, or the recent FEAST methods, as well as domain-specific iterative schemes relevant for the ESSEX quantum physics application. This report introduces the project structure and presents selected results which demonstrate the potential impact of ESSEX for efficient sparse solvers on highly scalable heterogeneous supercomputers.

1 Sparse Solvers for Exascale Computing

Energy-efficient execution, fault tolerance (FT), and exploiting extreme levels of parallelism of hierarchical and heterogeneous hardware structures are widely considered to be the basic requirements for application software to run on future exascale systems. Specific hardware structures and best programming models for the exascale systems are, however, not yet accessible, let alone settled. Thus, development of exascale applications can be considered as a research project on its own. Existing software structures, numerical methods, and conventional programming and optimization approaches need to be reconsidered. New techniques such as FT or parallel execution on heterogeneous hardware have to be developed.

A wide range of sparse linear algebra applications from quantum physics to fluid dynamics have already identified urgent problems which can only be solved with exascale computers. The relevant sparse linear solvers are typically based on iterative subspace methods, including advanced preconditioners. At the lowest level, large sparse matrix-vector multiplications (spMVM) and vector-vector operations are frequently the most time-consuming building blocks. Most of the available sparse linear (solver) packages were designed in the early 1990s for moderately parallel, homogeneous, and reliable computers (e.g., PETSc [1] or (P)ARPACK [2]) or with a strong focus on object orientation and abstraction (e.g., Anasazi [3]). Numerically intensive kernels are still encapsulated in independently developed libraries (see LAMA [4,5] for a recent project), which

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part II, LNCS 8806, pp. 577–588, 2014.
© Springer International Publishing Switzerland 2014

rules out the opportunity for coherent performance-aware and fault-tolerant co-design throughout all software layers up to and including the application. For the same reason this approach makes it difficult to accommodate new hardware architectures (see, e.g., the status of GPGPU support in PETSc [1]) and programming models, which is critical in view of the unknown shape of hardware and software environments for exascale systems. Autotuning approaches such as, e.g., pOSKI [6], try to relieve the developer from the tedious task of finding the problem- and hardware-specific optimization opportunities. While this may seem attractive, it does not generate true insight into the real performance issues, and shares the main problems of all encapsulated libraries.

These observations raise doubts about the fitness of existing sparse matrix applications for future exascale environments: (i) The problem of *optimal performance* and *energy efficiency* on highly parallel, heterogeneous node architectures is far from being solved. When the ESSEX project started, sparse data formats were strongly hardware-dependent, which was a major obstacle for software development and code efficiency on strongly heterogeneous systems. (ii) Existing sparse linear algebra frameworks use a strictly data-parallel approach, ignoring the need for additional levels of parallelism. These would allow for the concurrent execution of, e.g., several independent building blocks, asynchronous communication, or FT schemes. (iii) The standard solution for FT is classic synchronous file-based checkpoint/restart, which will lead to severe problems on exascale. Multi-level checkpointing [7] has recently been proposed as an alternative but it is not clear if those hierarchical disk systems will be affordable in terms of energy consumption at exascale. There is very little work on automatic FT approaches with minimal or no file system involvement beyond long-known "RAID-like" ideas [8].

The need for a complete re-design of existing large-scale application software with these exascale challenges in mind has been recognized by research activities in dense linear algebra [9]. The sparse linear algebra community still lacks such an initiative, in particular with respect to the co-design of all software layers, including basic building blocks, numerical methods, and application layers. Focusing on sparse linear eigenproblems from quantum physics, the ESSEX project is an attempt to close this gap. It will deliver methods and programming techniques that can serve as blueprints for other exascale initiatives in the sparse linear algebra community.

2 ESSEX Project Overview

The ESSEX project addresses the three fundamental software layers of computational science and engineering: basic building blocks, algorithms, and applications. The need for coherent FT approaches and energy efficiency are strongly integrating components which drive the tight exchange between the classic layers (see Fig. 1). Both vertical pillars share the important constraint of minimal time to solution. For a more detailed analysis of the relevance of code optimization for energy efficiency see [10]. Thus, the complete project is embedded in a structured holistic performance engineering process, which detects and guides performance potentials across the classic layers. This process is driven by the activities at the building blocks and successively integrates topics above them.

At the **application layer**, the ESSEX project is motivated by large-scale eigenvalue problems from quantum physics, including highly relevant application fields such as graphene and topological insulators. Determining the relevant static and dynamic physical properties requires addressing various aspects of an eigenvalue problem that involves extremely sparse matrices with dimensions between 10^9 and 10^{14}: The computation of (i) the minimal and the maximal eigenvalue, (ii) a block of eigenpairs at the lower end or at the middle of the spectrum, and (iii) high quality approximations to the complete eigenvalue spectrum. All these aspects are of general interest, and not restricted to the applications considered in this project.

The **algorithms layer** has identified appropriate numerical schemes to determine blocks of eigenpairs including both classic schemes (Lanczos and Jacobi-Davidson [JADA]) with relevant preconditioners and the recently introduced FEAST [11] algorithm. The kernel polynomial method (KPM) [12] and related polynomial expansion schemes (ChebTP [13,14],CFET [15]) are employed to compute the density of states, excitation spectra, and dynamical properties.

Figure 2 demonstrates how the numerical methods in ESSEX map to the physical properties to be computed. Enabling these popular algorithms for exascale is of broad interest. Even the KPM, which has been application-specific for quantum physics and chemistry for a long time, has recently gained wider attention [16,17].

The **basic building block layer** provides a collection of all relevant basic operations (such as parallel spMVM, vector-vector operations, and global reductions) and efficient FT strategies, all tailored to the needs of the other two layers. The major design goals for these building blocks are: (i) "Optimal" performance, in the sense that a suitable performance model is available that describes the relevant execution bottlenecks, and that the implementation operates at these bottlenecks. (ii) Minimum impact of FT overhead on time to solution.

Although there is a huge variety of potential programming models to choose from, the project consistently follows an "MPI+X" approach, where "X" addresses parallelism on the compute node level, be it multiple cores or accelerators. CUDA, OpenMP, and POSIX threads are typical choices for "X" in our project.

The major challenges addressed at this layer are, e.g., developing optimized data structures for all available hardware architectures, obtaining high parallel performance when executing on heterogeneous compute nodes (using standard CPUs, GPGPUs, and Intel Xeon Phi concurrently), or hiding the costs of FT schemes based on checkpoint-restart strategies. Performance engineering, used as a well-defined process targeting

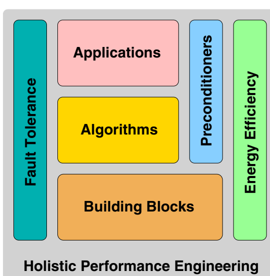

Fig. 1. Basic structure of the ESSEX project. The vertical activities are driven by a holistic performance engineering process and span the classic boundaries of application, algorithms and basic building blocks.

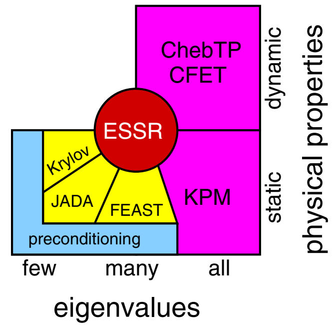

Fig. 2. The ESSEX research addresses the eigenproblem with classic and new eigensolvers (Krylov, JADA, FEAST) and preconditioners, established Chebyshev techniques (KPM, ChebTP) and novel implementations (CFET). The implemented methods will be part of the Exascale Sparse Solver Repository (ESSR).

"optimal" performance, starts at the basic blocks and is instrumental for developing insights into the relevant performance-limiting bottlenecks. Since it extends into the algorithms and application layers, it breaks up abstraction boundaries and enables optimizations that would be impossible in a pure library-based approach, where building blocks and algorithms are abstracted and inaccessibly wrapped in libraries.

The developments of all layers will eventually contribute to the Exascale Sparse Solver Repository (ESSR), which will become publicly available.

3 Results and Work in Progress

This section presents selected results and current work in progress. The topics have been chosen so as to demonstrate the broad range of activities and the potential general impact of the ESSEX project.

3.1 Applications

Quantum physics and quantum chemistry applications rely on a variety of numerical linear algebra techniques. Coming from the application side we can broadly classify the possible algorithmic choices by whether only a few eigenvalues are needed—such as for the computation of low-energy properties or ground states—, or whether all eigenvalues contribute—such as for the computation of spectral functions or dynamical properties (see Fig. 2).

To illustrate this concept we briefly develop the central computational ideas underlying one particular application scenario, the computation of the electronic properties of graphene samples [18,19]. At the core of the computation are energy-resolved functions

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

of an observable X. Here, H is the matrix representation of the physical Hamilton operator, with N eigenvalues E_n and eigenstates ψ_n. In this particular expression, all matrices are symmetric (or Hermitian). Physical quantities such as the electric conductivity are now obtained as a weighted mean of the form $\int X(\omega)f(\omega)d\omega$, where $f(\omega)$ is a prescribed scalar function such as the Boltzmann or Fermi-Dirac weight. In the special

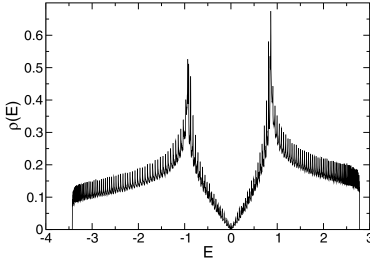

Fig. 3. Density of states (DOS) of a graphene nano-ribbon [20], computed with the KPM-DOS method (see Alg. 1)

case $X = I$ and $f(\omega) \equiv 1$, the above function gives the density of states (DOS), which counts the number of eigenvalues in a given interval (see Fig. 3).

3.2 Algorithms

In (1) all eigenpairs of H contribute, but explicit computation of a substantial fraction or even of all eigenpairs is not feasible. It is now the application that further dictates the algorithmic choice.

FEAST algorithm For very narrow $f(\omega)$, which occurs for instance at low temperatures, we can compute the eigenpairs selected by $f(\omega)$ with the FEAST algorithm. Typically about 200 to 400 eigenpairs are requested. FEAST has not yet reached the algorithmic maturity of JADA and other well-established iterative eigensolvers (cf., e.g., [21]). Therefore, performance optimization for FEAST must be accompanied, and preceded, by enhancements of the basic scheme in order to improve its robustness and numerical efficiency. Recent methodological progress and first numerical results for graphene nano-ribbon models are reported in [20,22].

Chebyshev polynomial expansion schemes If more eigenvalues contribute in the sum (1) for broader $f(\omega)$ we compute a polynomial approximation to the entire function $X(\omega)$ with the KPM. In this way, explicit computation of eigenpairs can be avoided. The KPM is based on the recurrence relation

$$|v_0\rangle = |v\rangle \; , |v_1\rangle = \tilde{H}|v_0\rangle \; , |v_{m+1}\rangle = 2\tilde{H}|v_m\rangle - |v_{m-1}\rangle \tag{2}$$

for vectors $|v_m\rangle = T_m[\tilde{H}]|v\rangle$, where the $T_m(x)$ are the Chebyshev polynomials of the first kind. Note that the original matrix has been rescaled to $\tilde{H} = a(H-b)$ such that all eigenvalues lie in the definition range $[-1,1]$ of the $T_m(x)$. To this end, an approximation to the minimum and maximum eigenvalues is computed initially, for which the Lanczos algorithm can be used. The corresponding Chebyshev moments $\mu_m = \int X(\omega)T_m(\omega)d\omega$ of $X(\omega)$ are obtained from the scalar products

$$\mu_m = \langle v|T_m(\tilde{H})X|v\rangle = \langle v_m|X|v_0\rangle . \tag{3}$$

From these moments, the function $X(\omega)$ is reconstructed as a Fourier transform. The full trace tr[...] in (1) is replaced by a stochastic sum over several random starting vectors $|v\rangle$. For more details see our KPM review [12].

Several computational steps can now be identified in the above scheme: spMVM, vector-vector operations, scalar products, and an outer loop over random vectors.

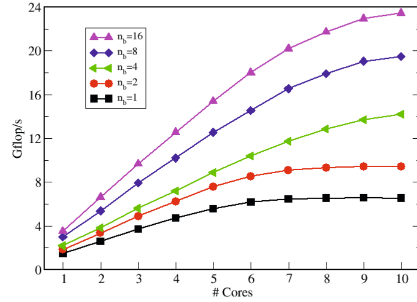

Fig. 4. Performance of block spMVM for various numbers of vectors (n_b) involved in the vector block. Measurements have been performed on a single Intel Xeon E5-2660 v2 processor (fixed clock speed of 2.2 GHz). The matrix has approximately 10^7 rows and an average number of non-zero entries per row of $N_{nzr} = 14$.

A straightforward implementation of these steps leads to the KPM-DOS algorithm discussed below (Alg. 1). Performance engineering, which exploits the specific combination in which the individual computational steps occur together with the different levels of parallelism, results in a highly efficient algorithm (cf. Sect. 3.4) that is tailored to achieve best performance for the KPM-DOS application class represented by Fig. 3.

In this way application-specific information enters at all stages of the development cycle, which is characteristic for the strong vertical integration that we pursue in the ESSEX project. It applies equally to the other applications and algorithms addressed. For instance in the graphene application, specifically in the computation of time-resolved electron dynamics, the above FEAST/KPM steps are complemented by computations of the matrix exponential $e^{-\mathrm{i}Ht}$, for which we use again Chebyshev techniques.

Parallel block JADA Many quantum physics applications, such as strongly correlated systems, require the computation of a few extremal eigenvalues of a symmetric matrix, for which we use the classic JADA algorithm. The implementation of iterative JADA solvers relies on spMVM and (block) vector-vector operations. Hence, a functional interface to the basic building blocks library GHOST (see Sect. 3.3) has been developed. In order to increase the impact of our new JADA implementation, we also implement this interface for other linear algebra packages such as the Trilinos[1] project. On the other hand, the abstraction introduced here allows us to exploit the work of others and makes GHOST compatible with, e.g., the "tall skinny QR" factorization (TSQR), block Krylov-Schur and communication-avoiding GMRES in Trilinos.

Two JADA variants have been implemented: A single-vector method as a reference solver and an experimental pipelined block method that performs as key operations a block spMVM (spMVM applied to multiple vectors) and a block orthogonalization step [23]. Block spMVM reduces overall main memory data traffic as compared to an equivalent series of standard spMVMs. Using an highly optimized block spMVM routine (from the GHOST library) based on a row-wise storage scheme for the vector block, a performance gain of almost four can be typically achieved on a full socket basis (cf. Fig. 4). In a set of representative experiments this advantage outweighed the increase in floating point operations due to the blocked JADA algorithm in almost all cases so that an overall speed-up of blocked JADA of around 50% is achieved on the socket level for recent Intel processors. The performance advantage continues into the parallel region as demonstrated by first measurements using a moderately sized test

[1] `http://trilinos.sandia.gov`

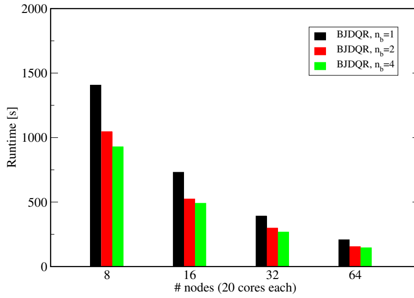

Fig. 5. Hybrid-parallel (MPI+OpenMP) execution times of the block JADA solver for up to 64 nodes (1280 cores) of a Infiniband cluster using the compute nodes specified in Fig. 4, i.e. each node carries 20 physical cores. The matrix has about 1.6×10^8 rows and 2.6×10^9 non-zero entries.

matrix (see Fig. 5). Note that in the block variant the average message size increases (at constant overall communication volume) and we have furthermore eliminated global synchronization points wherever possible.

In the current package, JADA uses a pipelined block GMRES method without further preconditioning for the solution of the correction equation. In the future we will integrate advanced preconditioning techniques to accelerate convergence. Furthermore, algorithmic overlapping of communication and computation will be made possible by exploiting the GHOST task queueing system, which will enable, e.g., overlapping spMVM communication with numerical operations in other JADA or GMRES loops. Another focus of future work will be to include GPGPUs in all JADA operations, which is already possible with GHOST, but not fully implemented in our interface.

3.3 Basic Building Blocks

As a first step towards a flexible repository of basic building blocks, multi-threaded low-level routines for basic operations such as spMVM, vector-vector operations, etc., were developed. Naturally, the spMVM has received special attention since it is the hot spot in most of the algorithms employed in the project. OpenMP and CUDA were chosen as the “X” programming model in order to address the most popular computing devices in modern, heterogeneous clusters. On the distributed-memory level, the MPI implementation allows for a simple MPI-only model as well as for hybrid approaches where each process owns multiple threads, possibly dedicated to the separate tasks of communication and computation. This makes it possible to achieve an explicit overlap between computation and communication, even if the underlying MPI implementation does not support truly asynchronous point-to-point transfers for large messages.

The FT aspect of the building blocks layer was initially addressed by an implementation of checkpointing for a lattice-Boltzmann flow solver using dedicated checkpoint threads [24], by which we could demonstrate the feasibility of asynchronous checkpointing and its low overhead on modern commodity systems. In order to get a more complete view of available checkpointing techniques, several existing solutions were investigated and compared [25,26]. However, checkpoint/restart is only the most basic FT technique. Future systems will not be able to sustain the continuous I/O load caused by checkpoint/restart when the job-level mean time between failure is of the order of minutes. Hence, research is going on in many directions in search for fault-tolerant programming models which enable applications to continue running even if a

Listing 1.1. Spawning a multi-threaded computation and a single-threaded checkpointing task using GHOST.

```
// define task: checkpointing with 1 thread
ghost_task_create(&chkpTask, 1, curTask->LD, &chkp_func, \
      (void *)&chkp_func_args, GHOST_TASK_DEFAULT, NULL, 0);
// define task: compute with N-1 threads
ghost_task_create(&compTask, curTask->nThreads-1, \
      curTask->LD, &comp_func, (void *)&comp_func_args, \
      GHOST_TASK_DEFAULT, NULL, 0);
// initiate tasks
ghost_task_enqueue(chkpTask); ghost_task_enqueue(compTask);
// wait for completion
ghost_task_wait(chkpTask); ghost_task_wait(compTask);
```

node fails. Since the MPI standard does not yet contain any such features today, we have first ported a distributed-memory spMVM operation to GPI [27]. GPI[2] is an open source implementation of the GASPI PGAS standard, and explicitly supports continuous execution after hardware failures. Work is ongoing to test these facilities using the KPM-DOS application.

Taking as much complexity as possible out of the developer's hands without sacrificing full control over performance and execution modes (such as affinity, threading, functional parallelism) were conflicting goals in the development of the basic blocks layer. We have addressed this challenge by developing GHOST (General Hybrid Optimized Sparse Toolkit). GHOST is a library that can be used from C/C++ and Fortran programs. It implements a flexible thread-tasking model on the process level, providing the required affinity and resource management functions to support functional parallelism as needed by all project layers. For instance, a background task for parallel checkpointing can be initiated with a single function call, while another task is executing a sparse MVM (see Listing 1.1).

Addressing heterogeneity, especially when dealing with sparse matrices, requires more than a proper choice of programming model. The optimal format for storing sparse matrices was, up until recently, highly hardware-dependent: On standard cache-based processors the compressed row storage (CRS) format usually leads to best performance, while GPGPUs require the fundamentally different ELLPACK or one of its derivatives [28]. On vector computers, the jagged diagonals storage (JDS) is most suitable since it leads to long, easily vectorizable inner loops, while the optimal format for the new Intel Xeon Phi architecture was yet to be found. In the basic building blocks layer we have developed SELL-C-σ, a sparse matrix storage format that yields best or competitive performance on all modern computer architectures (see Fig. 6), with the added benefit of saving memory compared to the popular ELLPACK-based variants on GPGPUs [29]. This format facilitates the programming of heterogeneous hardware, since

[2] http://www.gpi-site.com

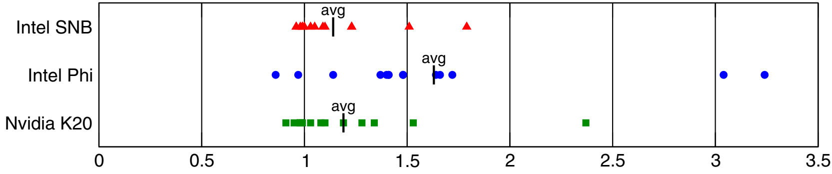

Fig. 6. Relative performance benefit of the unified SELL-32-σ format over the vendor-supplied library spMVM performance for twelve "non-pathological" test cases in the UoF matrix collection (see [29] for details) on Intel Sandy Bridge ("SNB"), Intel Xeon Phi, and Nvidia K20. A format similar to SELL-C-σ will be supported in a future release of the Intel Math Kernel Library [32].

load balancing and FT features do not have to take format conversions into account. Furthermore it will greatly ease the development of efficient code on upcoming unified memory architectures, where the host CPU and the accelerator hardware share memory. SELL-C-σ has immediately been taken up and adapted to special needs by several research groups [30,31].

3.4 Holistic Performance Engineering

The performance optimization process applied to the computation of the DOS with the KPM (KPM-DOS, for $X = I$ in (1)) is a simple but very instructive example for the advantages of a holistic view on the complete software stack.

A standard scheme for computing the Chebyshev moments $\{\mu_m; m = 0, \ldots, M\}$ for a given M is shown in Alg. 1 (middle loop over m). In terms of computational complexity the relevant step is the construction of the vectors $|v_m\rangle = T_m[\tilde{H}]|v\rangle$ through the recurrence (2). Note that, using the relation $T_{m+n}(x) = 2T_m(x)T_n(x) - T_{m-n}(x)$, the algorithm can be formulated as presented, delivering two moments ($\mu_{2m} = 2\eta_{2m} - \mu_0, \mu_{2m+1} = 2\eta_{2m+1} - \mu_1$) per spMVM operation.

The Chebyshev scheme requires a spMVM routine involving the original matrix H and various vector-vector operations including a scalar product as basic building blocks. Typically, highly optimized subroutines are provided by an external low-level library and are called in the order shown. As a consequence, besides the spMVM, eight vectors of matrix dimension have to be loaded and four stored from/to main memory, generating data traffic which can be as high as in the spMVM alone. Extending the optimization scope to the algorithmic layer allows to define a tailored spMVM routine that eliminates all data transfers for the vector-vector operations. Those operations are performed in the spMVM step when the relevant data is available in registers or in the L1 cache. Thus, the overall data traffic is reduced to a single basic spMVM step (see Alg. 2). Further performance potential becomes accessible if the optimization scope also includes the application problem, which is the KPM-DOS computation. Here, the outer loop runs over a set of random vectors for which the Chebyshev moments are computed independently, loading the full matrix in each iteration. Applying the tailored spMVM to a block of random vectors can add the substantial performance gains demonstrated in Fig. 4 to our application scenario. The optimal number of vectors in the block is set by

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

Algorithm 1: Basic scheme to compute the Chebyshev moments (KPM-DOS) for a set of R random vectors $\{|\text{rand}()\rangle\}$ using the standard spMVM operation

a subtle interplay of matrix dimension, matrix bandwidth, and cache size, and is subject to current research in ESSEX. For the benchmarks presented below, eight vectors per block are chosen, which reduces the overall data traffic for loading matrix information accordingly. Note that the use of block vectors is only possible if KPM is applied to compute the density of states. If a static excitation spectrum is determined there is no outer loop in the scheme.

For the test matrix and a single socket of the compute node used in Fig. 4, the two successive optimizations have improved the performance from 5.5 GF/s (basic version) to 8.3 GF/s (tailored spMVM) to finally 21.6 GF/s (blocked tailored spMVM). Though the matrix is rather small, the KPM scheme is still completely memory bound. Hence, considering all software layers in the optimization process results in an almost 4× speed-up. Note that in the basic version each of the different subroutines had been individually well optimized: The basic spMVM step runs at a performance of 6.5 GF/s, indicating a very good utilization of the memory bandwidth bottleneck (45 GB/s read-only bandwidth for the test system) according to the spMVM performance model presented in [29].

for $r = 0$ *to* $R-1$ **do**
  $|v\rangle = |\text{rand}()\rangle$;
  $|w\rangle = a(H-b)|v\rangle$ & $\mu_0 = \langle v|v\rangle$ & $\mu_1 = \langle w|v\rangle$;
  for $m = 1$ *to* $M/2$ **do**
    swap($|w\rangle, |v\rangle$);
    $|w\rangle = 2a(H-b)|v\rangle - |w\rangle$ & $\eta_{2m} = \langle v|v\rangle$ & $\eta_{2m+1} = \langle w|v\rangle$;
  end
end

Algorithm 2: Improved computation of Chebyshev moments (KPM-DOS) with a tailored spMVM operation. Operations chained by “&” in a single line do not cause main memory traffic as they are performed in the spMVM operation.

4 Conclusions

In the first 18 months the ESSEX project has made substantial contributions to the sparse linear algebra community reaching far beyond its application area. Other groups have already picked up several results, and new collaborations with projects both within SPPEXA and beyond have been established. A preliminary version of the Exascale Sparse Solver Repository (ESSR) will be released by the end of 2014.

Acknowledgments. This work is supported by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA) under project ESSEX.

References

1. Threading and GPGPU support in PETSc, http://www.mcs.anl.gov/petsc/features/
2. Parallel Arnoldi package (PARPACK) homepage, http://www.caam.rice.edu/~kristyn/parpack_home.html
3. Anasazi package homepage, http://trilinos.sandia.gov/packages/anasazi/
4. LAMA — Library for Accelerated Math Applications, http://www.libama.org
5. Förster, M., Kraus, J.: Scalable parallel AMG on ccNUMA machines with OpenMP. Computer Science - Research and Development 26, 221–228 (2011) ISSN 1865-2034
6. pOSKI: parallel optimized sparse kernel interface, http://bebop.cs.berkeley.edu/poski
7. Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N., Matsuoka, S.: FTI: high performance fault tolerance interface for hybrid systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2011, pp. 32:1–32:32. ACM, New York (2011)
8. Plank, J.S., Kim, Y., Dongarra, J.J.: Algorithm-based diskless checkpointing for fault-tolerant matrix operations. In: Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing, FTCS 1995, pp. 351–360. IEEE Computer Society, Washington, DC (1995)
9. Horton, M., Tomov, S., Dongarra, J.: A class of hybrid LAPACK algorithms for multicore and GPU architectures. In: Symposium on Application Accelerators in High-Performance Computing, pp. 150–158. IEEE Computer Society, Los Alamitos (2011)
10. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multicore chips via simple machine models. Concurrency Computat. Pract. Exper. (2013), doi:10.1002/cpe.3180
11. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Phys. Rev. B 79, 115112 (2009)
12. Weiße, A., Wellein, G., Alvermann, A., Fehske, H.: The kernel polynomial method. Rev. Mod. Phys. 78, 275 (2006)
13. Tal-Ezer, H., Kosloff, R.: An accurate and efficient scheme for propagating the time dependent Schrödinger equation. J. Chem. Phys. 81, 3967 (1984)
14. Fehske, H., Schleede, J., Schubert, G., Wellein, G., Filinov, V.S., Bishop, A.R.: Numerical approaches to time evolution of complex quantum systems. Phys. Lett. A 373, 2182 (2009)
15. Alvermann, A., Fehske, H.: High-order commutator-free exponential time-propagation of driven quantum systems. J. Comp. Phys. 230, 5930 (2011)
16. di Napoli, E., Polizzi, E., Saad, Y.: Efficient estimation of eigenvalue counts in an interval, Preprint arXiv:1308.4275 (2013)

17. Bhardwaj, O., Ineichen, Y., Bekas, C., Curioni, A.: Highly scalable linear time estimation of spectrograms - a tool for very large scale data analysis. Poster at 2013 ACM/IEEE International Conference on High Performance Computing Networking, Storage and Analysis (2013)
18. Pieper, A., Schubert, G., Wellein, G., Fehske, H.: Effects of disorder and contacts on transport through graphene nanoribbons. Phys. Rev. B 88, 195409 (2013)
19. Pieper, A., Heinisch, R.L., Wellein, G., Fehske, H.: Dot-bound and dispersive states in graphene quantum dot superlattices. Phys. Rev. B 89, 165121 (2014)
20. Krämer, L., Galgon, M., Lang, B., Alvermann, A., Fehske, H., Pieper, A.: Improving robustness of the FEAST algorithm and solving eigenvalue problems from graphene nanoribbons (Submitted to PAMM 2014)
21. Krämer, L., Di Napoli, E., Galgon, M., Lang, B., Bientinesi, P.: Dissecting the FEAST algorithm for generalized eigenproblems. J. Comput. Appl. Math. 244, 1–9 (2013)
22. Krämer, L.: Integration Based Solvers for Standard and Generalized Eigenvalue Problems. Ph.D. thesis, Bergische Universität Wuppertal (2014)
23. Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H.: Increasing the performance of the Jacobi-Davidson method by blocking. SIAM J. Sci. Comput. (Submitted)
24. Shahzad, F., Wittmann, M., Zeiser, T., Wellein, G.: Asynchronous checkpointing by dedicated checkpoint threads. In: Träff, J.L., Benkner, S., Dongarra, J.J. (eds.) EuroMPI 2012. LNCS, vol. 7490, pp. 289–290. Springer, Heidelberg (2012)
25. Shahzad, F., Wittmann, M., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: A survey of checkpoint/restart techniques on distributed memory systems. Parallel Processing Letters 23(04), 13400111–134001120 (2013)
26. Shahzad, F., Wittmann, M., Zeiser, T., Hager, G., Wellein, G.: An evaluation of different I/O techniques for checkpoint/restart. In: Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), pp. 1708–1716. IEEE Computer Society (2013)
27. Shahzad, F., Wittmann, M., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: PGAS implementation of SPMVM and LBM with GPI. In: Proceedings of the 7th International Conference on PGAS Programming Models, pp. 172–184 (2013)
28. Bell, N., Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In:Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 2009, pp. 18:1–18:11. ACM, New York (2009)
29. Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.: 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)
30. Müthing, S., Ribbrock, D., Göddeke, D.: Integrating multi-threading and accelerators into DUNE-ISTL. In: Proceedings of ENUMATH 2013 (accepted 2014)
31. Anzt, H., Tomov, S., Dongarra, J.: Implementing a sparse matrix vector product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs. Tech. rep. (March 2014), http://www.eecs.utk.edu/resources/library/585
32. Intel Math Kernel Library (MKL), https://software.intel.com/en-us/intel-mkl