Jacobi-Davidson algorithm with fast matrix-vector multiplication on massively parallel and vector supercomputers

M. Kinateder[1], G. Wellein[2], A. Basermann[3] and H. Fehske[1]

[1] Physikalisches Institut, Universität Bayreuth, D-95440 Bayreuth
[2] Regionales Rechenzentrum Erlangen, Universität Erlangen, D-91058 Erlangen
[3] C&C Research Laboratories, NEC Europe Ltd, D-53757 Sankt Augustin

Abstract. The exact diagonalization of very large sparse matrices is a numerical problem common to various fields in science and engineering. We present an advanced eigenvalue alorithm - the so-called Jacobi-Davidson algorithm - in combination with an efficient parallel matrix-vector multiplication. This implementation allows the calculation of several specified eigenvalues with high accuracy on modern supercomputers, such as CRAY T3E and NEC SX-4. Exemplarily the numerical technique is applied to analyze the ground state and spectral properties of the three-quarter filled Peierls-Hubbard Hamiltonian in relation to recent resonant Raman experiments on MX chain [-PtCl-] complexes.

1 Introduction

Many problems in theoretical physics are related to eigenvalue problems involving large sparse matrices. To investigate the low energy physics in the framework of the corresponding microscopic models, iterative subspace methods like the Lanczos algorithm [1] or the Davidson algorithm [2] are commonly used to calculate the ground state and some excited eigenstates. However these methods show a poor convergence and stability if the eigenvalues to be computed are not well separated or even degenerate. In that case, more advanced methods like the Jacobi-Davidson (JD) algorithm with preconditioning techniques [3,4] have to be used, providing both high resolution and rapid convergence.

In general, the computational requirements of these eigenvalue algorithms are determined by a matrix-vector multiplication (MVM) involving the large sparse Hamiltonian matrix. Thus the efficient use of modern supercomputers strongly depends on a parallel, fast and memory saving implementation of the matrix-vector multiplication. Considering the sparsity of the matrix, storage techniques like the compressed row storage (CRS) format [5] or the jagged diagonals storage (JDS) format [5] are used to store the non-zero elements of the matrix. Using these storage formats, both the memory and cpu-time requirements of the eigenvalue algorithms scale linearly with the matrix dimension.

A typical physical problem that requires the application of high-resolution techniques - such as JD - is the calculation of the eigenvalue spectrum for strongly interacting electron/spin-phonon systems. As an example, in section 5 we investigate the 1D Peierls-Hubbard Hamiltonian which serves as a generic model for the intensely studied MX-chain compounds. Most notably, in the strong-coupling regime we find clear signatures of local lattice distortions accompanied by intrinsic multi-phonon localization. Moreover, we are able to reproduce the observed red shift of the overtone resonance Raman spectrum.

The paper is organized as follows: In section 2 the JD algorithm is briefly introduced. Section 3 outlines the parallel implementation of the MVM using the JDS storage format. The accompanying performance analysis for modern supercomputers such as NEC SX-4 and CRAY T3E is given in section 4. Section 5 presents selected results of our MX-chain investigations.

2 Jacobi-Davidson algorithm with preconditioning

To solve large sparse Hermitian eigenvalue problems numerically, variants of a method proposed by Davidson [6] are frequently applied. These solvers use a succession of subspaces where the update of the subspace exploits approximate inverses of the problem matrix, A. For A, $A = A^H$ or $A^* = A^T$ holds where A^* denotes A with complex conjugate elements and $A^H = (A^T)^*$ (transposed and complex conjugate).

The basic idea is: Let $\mathbf{V}^k$ be a subspace of $\mathbb{R}^n$ with an orthonormal basis $w_1^k, \ldots, w_m^k$ and W the matrix with columns w_j^k, $S := W^H A W$, $\bar{\lambda}_j^k$ the eigenvalues of S, and T a matrix with the eigenvectors of S as columns. The columns x_j^k of WT are approximations to eigenvectors of A with Ritz values $\bar{\lambda}_j^k = (x_j^k)^H A x_j^k$ that approximate eigenvalues of A. Let us assume that $\bar{\lambda}_{j_s}^k, \ldots, \bar{\lambda}_{j_{s+l-1}}^k \in [\lambda_{\text{lower}}, \lambda_{\text{upper}}]$. For $j \in j_s, \ldots, j_{s+l-1}$ define

$$q_j^k = (A - \bar{\lambda}_j^k I)\, x_j^k\,, \qquad r_j^k = (\bar{A} - \bar{\lambda}_j^k I)^{-1} q_j^k\,, \tag{1}$$

and $\mathbf{V}^{k+1} = \operatorname{span}(\mathbf{V}^k \cup r_{j_s}^k \cup \ldots \cup r_{j_{s+l-1}}^k)$ where $\bar{A}$ is an easy to invert approximation to A ($\bar{A} = \operatorname{diag}(A)$ in [6]). Then $\mathbf{V}^{k+1}$ is an $(m+l)$-dimensional subspace of $\mathbb{R}^n$, and the repetition of the procedure above gives in general improved approximations to eigenvalues and -vectors. Restarting may increase efficiency.

For good convergence, $\mathbf{V}^k$ has to contain crude approximations to all eigenvectors of A with eigenvalues smaller than λ_{lower} [6]. The approximate inverse must not be too accurate, otherwise the method stalls. The reason for this was investigated in [3] and leads to the Jacobi-Davidson (JD) method with an improved definition of r_j^k:

$$[(I - x_j^k\,(x_j^k)^H)\,(\bar{A} - \bar{\lambda}_j^k I)\,(I - x_j^k\,(x_j^k)^H)]\, r_j^k = q_j^k\,. \tag{2}$$

The projection $(I - x_j^k (x_j^k)^H)$ in (2) is not easy to incorporate into the matrix, but there is no need to do so, and solving (2) is only slightly more expensive than solving (1).

The method converges quadratically for $\bar{A} = A$.

The character of the JD method is determined by the approximation $\bar{A}$ to A. For obtaining an approximate solution of the preconditioning system (2), we may try an iterative approach [3,7,8]. Here, a real symmetric version of the QMR algorithm is used that is directly applied to the projected system (2) with $\bar{A} = A$. The control of the QMR iteration is as follows. Iteration is stopped when the current residual norm is smaller than the residual norm of QMR in the previous inner JD iteration. By controlling the QMR residual norms, we achieve that the preconditioning system (2) is solved in low accuracy in the beginning and in increasing accuracy in the course of the JD iteration. For a block version of JD, the residual norms of each preconditioning system (2) are separately controlled for each eigenvector to approximate since some eigenvector approximations are more difficult to obtain than others. This adapts the control to the properties of the matrix's spectrum.

3 Implementation of matrix-vector multiplication

The efficiency of the Jacobi-Davidson method is determined by the performance of the matrix-vector multiplication which depends heavily on the storage scheme used. The extreme sparsity of the matrices used in this work ($\sim$ 10 nonzero elements per row, matrix dimension up to 10^8) calls for a solution where only the nonzero elements are stored. In previous works [8] a CRS format was implemented which has serious drawbacks on vector processors due to short vector lengths. To achieve high performance on all modern supercomputer architectures, a JDS scheme [5] was implemented.

3.1 The JDS format

The JDS (Jagged Diagonal Storage) format is a very general storage scheme for sparse matrices making no assumptions about the sparsity pattern. To illustrate the principles of the scheme, we introduce a 8×8 matrix A with nonzero elements a_{ij}. In a compression step, the zero elements are removed, nonzero elements are shifted to the left and the rows are made equal in length by padding them with trailing zeros:

$$A = \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & a_{36} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ 0 & 0 & 0 & a_{54} & a_{55} & 0 & a_{57} & 0 \\ 0 & 0 & 0 & a_{64} & 0 & a_{66} & a_{67} & 0 \\ 0 & 0 & 0 & a_{74} & a_{75} & a_{76} & a_{77} & 0 \\ 0 & 0 & 0 & a_{84} & 0 & 0 & 0 & a_{88} \end{bmatrix} \longrightarrow \begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & 0 \\ a_{22} & a_{23} & 0 & 0 & 0 & 0 \\ a_{32} & a_{33} & a_{34} & a_{36} & 0 & 0 \\ a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ a_{54} & a_{55} & a_{57} & 0 & 0 & 0 \\ a_{64} & a_{66} & a_{67} & 0 & 0 & 0 \\ a_{74} & a_{75} & a_{76} & a_{77} & 0 & 0 \\ a_{84} & a_{88} & 0 & 0 & 0 & 0 \end{bmatrix} . \quad (3)$$

To prevent storing theses trailing zeros, the rows are now reordered decreasingly according to the number of nonzeros per row:

$$A \longrightarrow \begin{bmatrix} a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ a_{74} & a_{75} & a_{76} & a_{77} & 0 & 0 \\ a_{32} & a_{33} & a_{34} & a_{36} & 0 & 0 \\ a_{54} & a_{55} & a_{57} & 0 & 0 & 0 \\ a_{64} & a_{66} & a_{67} & 0 & 0 & 0 \\ a_{22} & a_{23} & 0 & 0 & 0 & 0 \\ a_{84} & a_{88} & 0 & 0 & 0 & 0 \\ a_{11} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

The columns of the compressed and permuted matrix are called *Jagged Diagonals*. Obviously, the number of these *Jagged Diagonals* is equal to the maximum number `max_nz` of nonzeros per row. A suitable data structure to represent A in the JDS scheme is shown in Fig. 1. The `num_nz` nonzero elements of the compressed and permuted matrix are stored in a linear array `value(:)`, stringing the *Jagged Diagonals* together one by one. `col_ind(:)`,

```
JDS_MATRIX = record
        value   : array [1..num_nz] of REAL
        col_ind : array [1..num_nz] of INTEGER
        jd_ptr  : array [1..max_nz + 1] of INTEGER
        perm    : array [1..n] of INTEGER
end record
```

Fig. 1. Data structure of a $\mathtt{n} \times \mathtt{n}$ matrix in the JDS scheme

an array of the same length, contains the corresponding column indices. The pointer array `jd_ptr(:)` consists of the array indices belonging to the first elements of each *Jagged Diagonal* in `value(:)`. `jd_ptr(max_nz+1)` contains `num_nz+1`. The permutation information is stored in `perm(:)`, such that the original row number of the `i`-th row in (4) is `perm(i)`.

Fig. 2 shows the matrix A from (3) in the JDS format. The MVM (see Fig. 3) is performed along the *Jagged Diagonals*, providing an inner loop length essentially equal to the matrix dimension [1]. Increasing the loop length by performing the MVM along the *Jagged Diagonals* is the decisive factor concerning performance on vector processors [2].

[1] Of course, the inner loop length decreases with increasing process number when using parallel computers. The loop length is then of the order of matrix dimension divided by the number of employed processes.

[2] Note that using the CRS scheme yields an inner loop length of $\sim$ `max_nz` which hinders exploiting the performance-raising features of vector processors.

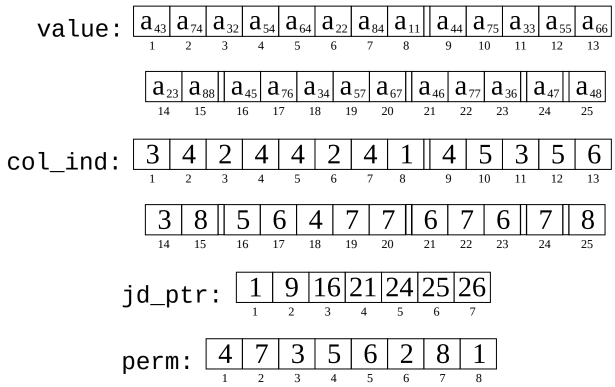

Fig. 2. Matrix A in the JDS scheme

```
for i = 1, ..., n do
    y(i) = 0
end for
for j = 1, .., max_nz do
    for i = 1, .., (jd_ptr(j + 1) − jd_ptr(j)) do
        y(i) = y(i) + value(jd_ptr(j) + i − 1) · x(col_ind(jd_ptr(j) + i − 1))
    end for
end for
```

Fig. 3. JDS matrix-vector multiplication $\mathbf{y} = A\mathbf{x}$

3.2 Communication Scheme

To keep memory requirements as low as possible on multiprocessor systems with distributed memory, the matrix data is spread over the processes involved in eigenvalue computation. Using the matrix A from (3) and a vector **x**, a proven data distribution is shown in Fig. 4. Matrix elements requiring

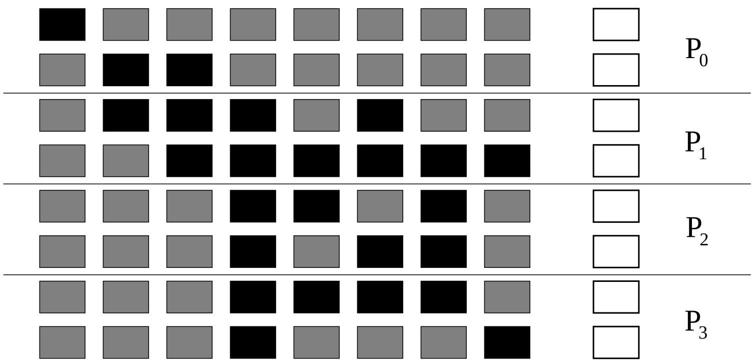

Fig. 4. Distribution of the matrix A and the vector $\mathbf{x}$ on 4 processes ($P_0 - P_3$). All nonzero matrix elements are black.

access to vector data stored on remote processes (*non-local* vector elements) cause communication. Since the matrix does not change during eigenvalue computation, a definite communication scheme is constructed once, allowing an efficient way of exchanging data in each MVM. Fig. 5 illustrates all necessary communication steps during one MVM. Table 1 shows a list of local [3] variables used to represent this communication scheme.

`recnum`	Number of processes from which vector elements are received
`recid(recnum)`	IDs of these processes
`elrec(recnum)`	Number of elements received from each of the `recnum` processes
`sendnum`	Number of processes to which vector elements are sent
`sendid(sendnum)`	IDs of these processes
`elsend(sendnum)`	Number of elements sent to each of the `sendnum` processes

Table 1. Variables representing the communication scheme

[3] The term "local" refers to variables which can have different values on different processes.

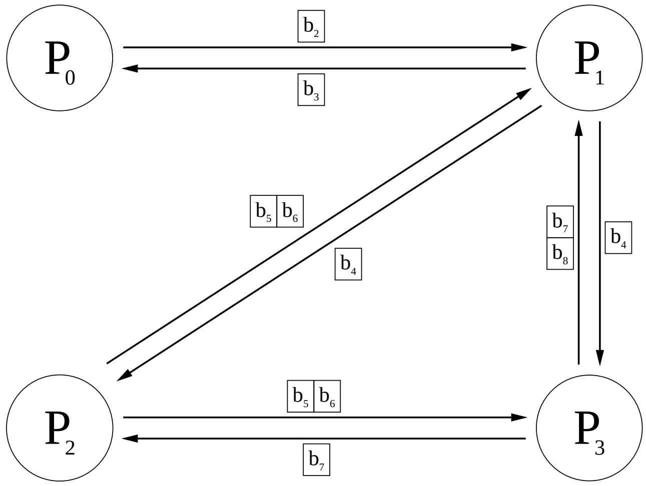

Fig. 5. Communication scheme resulting from the data distribution of Fig. 4

3.3 Matrix-Vector Multiplication

Since the matrix A from (3) is distributed row-wise among the processes (see Fig. 4) it can be locally transformed into the JDS format (see Fig. 6). Vectors involved in MVM automatically carry the same permuted order as the

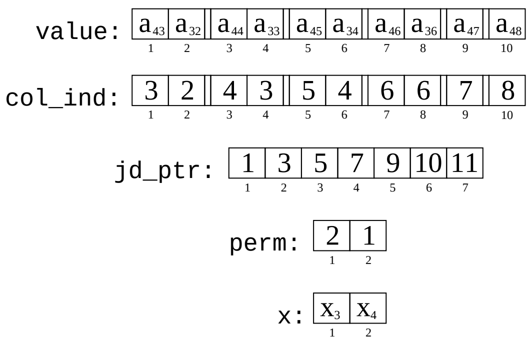

Fig. 6. Local portion of A and $\mathbf{x}$ on process P_1 in the JDS format

matrix rows. During MVM, each matrix element has to "find" the matching vector element it is multiplied with by using the permutation information together with the column indices. This information is incorporated into the matrix by permuting the column indices, applying a global permutation array `perm_global(:)`. This array is assembled out of the local permutation arrays `perm(:)` on every process.

To optimize vectorization, local and non-local parts of `x` are copied into a seperate array `recvec(:)` which serves as the "vector" in matrix-vector multiplication. Access to `recvec(:)` is controlled by two variables: `locp` contains the array index of the first local element, `nlocp(:)` contains the `recnum` array indices of the first vector elements received from each of the processes contained in `recid(:)`. To enable direct access to vector elements, the column indices in `col_ind(:)` are transformed into the matching array indices of `recvec(:)`. The whole transformation process of `col_ind(:)` is shown in Fig. 7.

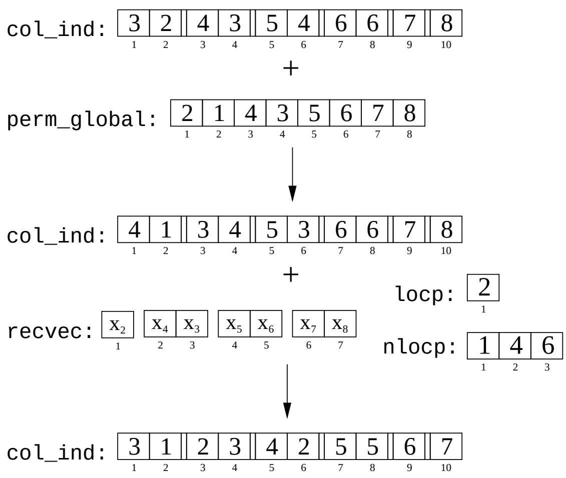

Fig. 7. Development of `col_ind(:)` on process P_1: First the column indices are permuted according to the matrix rows, applying the global permutation array `perm_global(:)`. Now, together with the data distribution, which is "known" on every process, the communication scheme can be constructed. In a second step, these permuted column indices are transformed to the matching array indices of `recvec`(:). Now, direct access to the correct local and non-local vector components during MVM is possible.

The *Jagged Diagonals* are processed as a whole during matrix-vector multiplication. Since *non-blocking* MPI routines are used for communication, it must be guaranteed that all non-local vector elements needed for processing a certain *Jagged Diagonal* are received and stored in `recvec(:)`. To provide an easy check, a bit pattern is created for each *Jagged Diagonal* during the construction of the communication scheme (see Fig. 8).

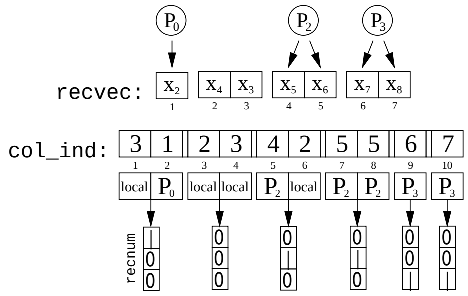

Fig. 8. Bit pattern derived from `recvec` and `col_ind` on process P_1 consisting of a bit array for each of the six *Jagged Diagonals*

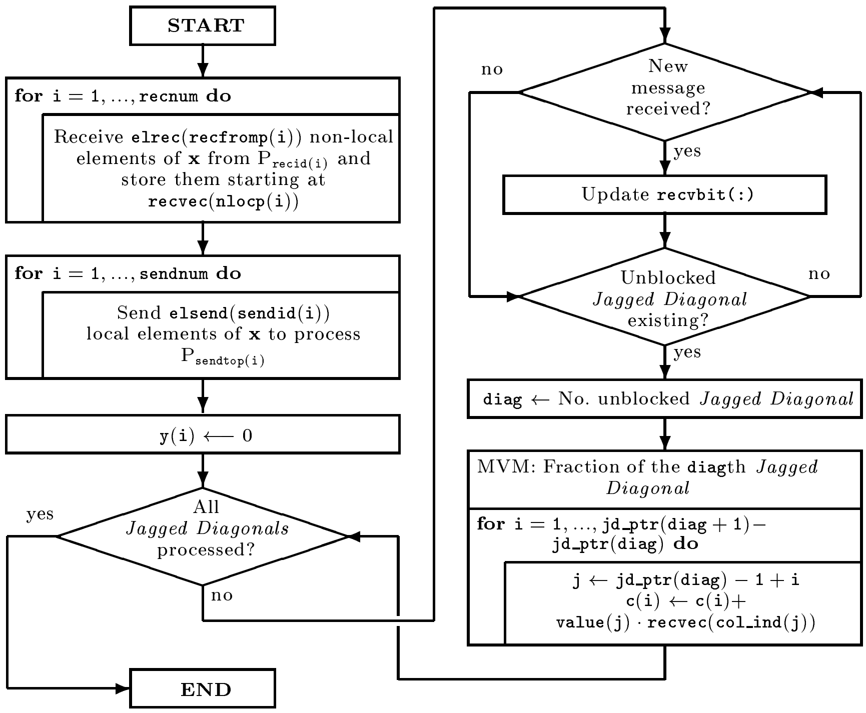

Fig. 9. JDS matrix-vector multiplication on process P_p

It contains "`|`" on position `i` if the receipt of data from process `recid(i)` is necessary before processing the *Jagged Diagonal* the bit pattern belongs to. During MVM a comparative bit pattern `recvbit(:)` is maintained, which is initialized with "`0`" and updated with "`|`" at position `i` if the message from `recid(i)` is received and stored. A *Jagged Diagonal* is blocked for processing until a bitwise AND-operation between its own and the comparative bit pattern leaves the own pattern unchanged. Fig. 9 gives an overview over the MVM.

4 Performance analysis

The JADA software package from C&C Research Laboratories, NEC Europe Ltd. with both the described new JDS based (cf. Section 3) and a standard CRS (cf. Ref [8]) based MVM implemtation bas been installed on massively parallel systems (CRAY T3E [HLRS, NIC Jülich]) and parallel vector computers (NEC SX-4 [HLRS], Fujitsu VPP700 [LRZ Munich]). To demonstrate the benefits of the JDS based implementation for both vector and RISC based architectures, an exact diagonalization of the model hamiltonian (6) introduced in section 5 has been performed at a fixed parameter set, obtaining the 20 lowest energy eigenvalues.[4] The characteristic parameters of the corresponding large sparse Hamiltonian matrix, the number of JD iterations and the number of MVMs are given in Tab. 2. Since the matrix generation routine is parallelized according to the basic principles of the PARPHON code introduced in Ref. [9,10], the maximum number of processes used is limited by the physical model parameters. In the case of the benchmark parameters presented in Tab. 2 a maximum of 16 processors can be used.

matrix dimension	263,536
number of nonzeros	1,904,120
nonzeros/zeros ratio	$2.7 \cdot 10^{-5}$
number of MVMs	7840
number of outer iterations	34
total iterations	1281
number of eigenvalues	20
accuracy ϵ of $\|\mathbf{r}\|$	10^{-7}

Table 2. Hamiltonian matrix and JD algorithm parameters used for the performance measurements

An overview over the total runtime for this benchmark based on both MVM implementations is presented in Fig. 11 using different number of processes (N_{pro}). Obivously, the JDS implementation performs much better than the CRS version on the Fujitsu VPP system. Of course this result will also be

[4] The CRAY T3E performance results were determined for the CRAY T3E-1200 system at NIC Jülich due to the larger amount of memory per node

valid for the NEC SX-4. As discussed above, this behaviour can be attributed to the different vector lengths of the inner loop of the MVM (cf. Fig. 3) which is determined by the matrix dimension for the JDS version (≈ 200.000) and by the number of nonzeros per row for the CRS version ($\approx 10-20$). Interestingly, there are no significant differences between JDS and CRS version for the RISC based CRAY T3E. Since the performance of the inner loop of the JDS based MVM is clearly limited by the memory access, it is evident that the NEC SX-4 runs faster than the Fujitsu VPP (cf. inset of Fig. 10).

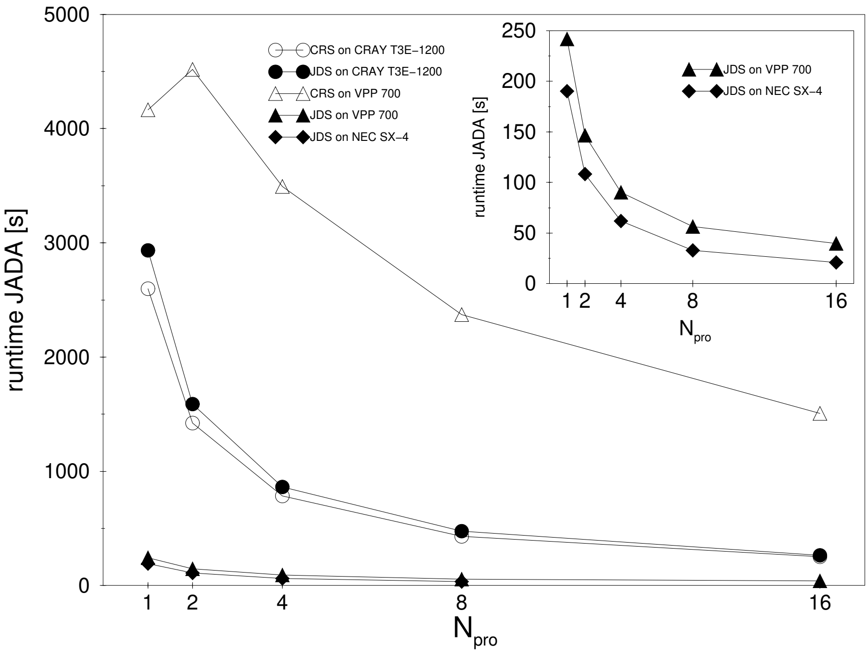

Fig. 10. Runtimes of the JD subroutine on different supercomputers. The inset shows the direct comparison between VPP 700 and NEC SX-4.

The same effect is shown in Tab. 3 where the total performance of the benchmark is given for different numbers of processes running the JDS based benchmark. Using one or two processes, the vector machines achieve more

	1	2	4	8	16
NEC SX-4	815	1431	2426	4526	7194
Fujitsu VPP 700	641	1056	1670	2648	3798
CRAYT3E-1200	53	97	174	315	570

Table 3. Total performance [MFlop/s] of the JDS based implementation using different numbers of processes

than 25% of the peak performance whereas the CRAY T3E only exploits 4% of the peak performance. Since the problem size is fixed, the utilization of the vector processor decreases with increasing N_{pro}, due to the communication overhead.

The parallel features of the implementation can be demonstrated in more detail by the parallel efficiency[5]

$$\eta_{pro} = \frac{T_1}{N_{pro} \times T_{N_{pro}}}, \tag{5}$$

which is depicted in Fig. 11(a). Both CRAY and NEC systems offer a satis-

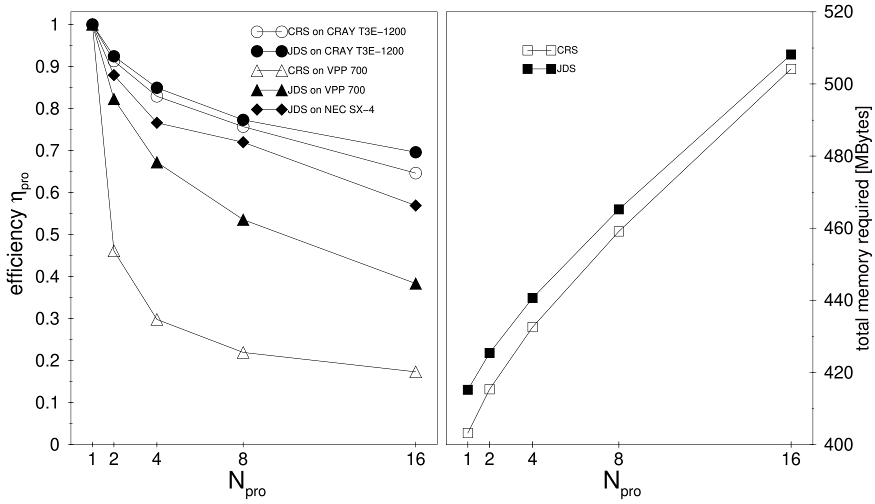

Fig. 11. Parallel efficiency (a) and total memory required (b) for JDS and CRS based implementation.

factory parallel efficency of more than 60% up to 16 processes. Surprisingly the parallel efficiency of the NEC SX-4 is - despite a higher communication bandwidth - slightly below that of the corresponding CRAY results. This suggests that the code is essentially latency- and not bandwidth-bound. In contrast to CRAY and NEC systems the benckmark offers a poor scalability on Fujitsu VPP. Whether these differences are caused by the communication scheme is still a subject under consideration.

Finally, Fig. 11(b) depicts the total memory requirements at fixed matrix size (cf. Tab 2) as function of processes involved in the computations.

[5] T_i sepcifies the runtime using i processes

There is a minor difference between CRS and JDS version concerning memory required. Its increase at higher process number is a phenomenon linked to bigger communication buffers, as with growing number of processes more vector components are "non-local".

5 Application: Intrinsic localized modes in quasi-1D CDW systems

Inspired by the recent observation of intrinsically localized vibrational modes in halide-bridged transition metal [-PtCl-] complexes [11], we study strong coupling effects between electronic and lattice degrees of freedom on the basis of a two-band, 3/4-filled Peierls-Hubbard model (PHM)

$$\begin{aligned}\mathcal{H} = &-t \sum_{\langle i,j\rangle\sigma} c_{i\sigma}^{\dagger} c_{j\sigma} + \sum_{i\sigma} \varepsilon_i n_{i\sigma} + \sum_i U_i n_{i\uparrow} n_{i\downarrow} \\ &+\lambda_I\,(b_I + b_I^{\dagger})(n_1 + n_3 - n_2 - n_4) + \hbar\omega_I\, b_I^{\dagger} b_I \\ &+\lambda_R\,(b_R + b_R^{\dagger})(n_2 - n_4) + \hbar\omega_R\, b_R^{\dagger} b_R\,. \end{aligned} \tag{6}$$

In (6), the $c_{i\sigma}^{[\dagger]}$ are fermion operators, $n_{i\sigma} = c_{i\sigma}^{\dagger} c_{i\sigma}$, and $b_R^{[\dagger]}$ and $b_I^{[\dagger]}$ are the boson operators for the Raman active (R) and infrared (I) optical phonon modes with bare phonon frequencies ω_R and ω_I. With regard to the quasi-1D charge density wave (CDW) system $\{[\mathrm{Pt(en)_2}][\mathrm{Pt(en)_2Cl_2}](\mathrm{ClO_4})_4\}$ (en= ethylenediamine), subsequently abbreviated as PtCl, the Pt (Cl) atoms are denoted by the site index $i = 2, 4$ ($i = 1, 3$). The CDW state is built up by alternating nominal Pt^{+4} and Pt^{+2} sites with a corresponding distortion of Cl^{-}-ions towards Pt^{+4}. To model the situation of a 3/4-filled charge transfer insulator within a *single-mode approach* (SMA), we only include the R-(ν_1)-mode with $\hbar\omega_R = 0.05$ [12], and parametrize the site energies by $\Delta = (\varepsilon_{\mathrm{Pt}} - \varepsilon_{\mathrm{Cl}})/t = 1.2$ and the Hubbard repulsions by $U_{\mathrm{Pt}} = 0.8$ and $U_{\mathrm{Cl}} = 0$ (all energies are measured in units of t). Alternatively, we employ a more realistic *double-mode approach* (DMA), using $\hbar\omega_I = 0.06$ for the I-(ν_2)-mode [12].

The ground-state and spectral properties of the four-site PHM are determined by finite-lattice diagonalization that preserves the full dynamics of the phonons [9].

Figure 12 shows the variation of the lowest energy states as a function of the electron-phonon coupling. In the weak-coupling region, the ground-state is basically a zero-phonon state (cf. Fig. 13 a) and the peaks in Fig. 12 (a) roughly correspond to multiples of the bare phonon frequency ω_R. As λ_R increases a strong mixing of electron and phonon degrees of freedom takes place, and finally the lowest states with total momentum $K = 0$ and $K = \pi$ become nearly degenerate. In this limit a non-linear lattice potential stabilizing so-called *intrinsically localized vibrational modes* (ILMs) is self-consistently generated [11]. Please note, that only the use of advanced algorithms such as

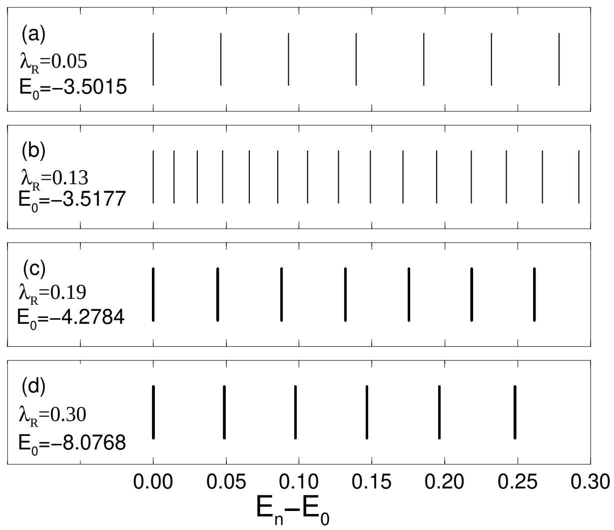

Fig. 12. Low-energy part of the eigenvalue spectrum of the PHM [periodic boundary conditions, SMA]. Eigenvalues in (c) and (d) are two-fold degenerate.

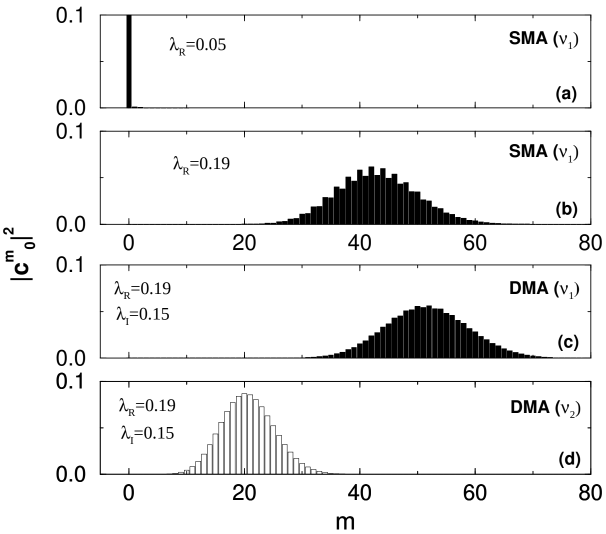

Fig. 13. Phonon distribution in the ground state for the antisymmetric I (white columns) and symmetric Cl-Pt-Cl stretch R (black columns) phonon modes.

the JD method enables a reliable computation of dense eigenvalue distributions including degenerate eigenvalues as presented in Fig. 12 (d).

A prominent feature of these bound states is their strong anharmonic redshift, resulting from the attractive interaction of Raman phonon quanta located at the same $PtCl_2$ unit. The calculated red shift, $r_n = [n\omega_R^{(1)} - \omega_R^{(n)}]/\omega_R^{(1)}$ with $\omega_R^{(n)} = (E_n - E_0)$, of the (doublet) overtones shown in Fig. 12 (c) is successfully compared to experimental data probed by resonance Raman scattering (see Tab. 4).

r_n [%]	$n=2$	3	4	5	5	7
$Pt^{37}Cl$ [1]	0.4	1.1	2.4	4.6	7.7	11.6
SMA	0.4	1.3	2.8	4.8	7.5	10.9

Table 4. Relative red shift of the lowest-energy peaks in dependence on the final quanta of the vibrational energy [$\lambda_R = 0.19$; SMA].

	SMA		DMA	$\Delta=0$
λ_I	0.0	0.0	0.15	0.15
λ_R	0.05	0.19	0.19	0.05
E_0	-3.5015	-4.2783	-5.3770	-3.2715
E_{kin}	-4.1115	-2.9186	-2.3202	-3.5985
E_{ph}	0.0001	2.1485	3.8490	0.2602
E_{el-ph}	-0.0036	-4.3026	-7.7017	-0.5433
E_U	0.6136	0.7945	0.7959	0.6102
$\langle n_{\mathrm{Cl}}\rangle$	1.7047	1.8834	1.9561	1.7070
$\langle n_{\mathrm{Pt}(2)}\rangle$	1.2953	0.2541	0.0949	1.2929
$\langle n_{\mathrm{Pt}(4)}\rangle$	1.2953	1.9791	1.9929	1.2929
$\langle n_1 n_2\rangle$	2.1053	0.3788	0.1441	2.1042
$\langle n_1 n_3\rangle$	2.8611	3.5396	3.8253	2.8705
$\langle n_2 n_4\rangle$	1.4990	0.4852	0.1824	1.4937
L_{Cl}	0.1897	0.0830	0.0323	0.1880
$L_{\mathrm{Pt}(2)}$	0.3962	0.1696	0.0682	0.3977
$L_{\mathrm{Pt}(4)}$	0.3962	0.0156	0.0053	0.3977
$S_s(\pi)$	0.1053	0.0524	0.0212	0.1042
Config.:	•-↑-•-↓	•-○-•-•	•-○-•-•	•-↑-•-↓

Table 5. Different energy contributions to E_0, local particle densities [$\langle n_i\rangle$], local magnetic moments [$L_i = 3\langle(n_{i\uparrow} - n_{i\downarrow})^2\rangle/4$], charge correlations [$\langle n_i n_j\rangle$] and magnetic structure factor [$S_s(\pi) = \frac{1}{N}\sum_{i,j}\langle S_i^z S_j^z\rangle \mathrm{e}^{i\pi(i-j)}$] in the ground-state of the PHM. Note that the effect of Δ can be obtained dynamically within the DMA (see column 4).

To elucidate the different nature of the ground state in the weak- and strong-coupling regimes, several characteristic quantities are listed in Tab. 5.

Obviously the self-localization transition is accompanied by significant changes in the spin- and charge correlations. As can be seen from the weight of the m-phonon state in the ground state, $|c_0^m|^2$ [9], depicted in Fig. 13, the appearance of the •-∘-•-• configuration is related to large occupation numbers of the localized vibrational (R) mode.

To summarize, the numerical results obtained for the PHM provide strong evidence for the existence of a dynamical spatial localization of vibrational energy (ILM) in MX solids, due to high intrinsic nonlinearity from strong electron lattice coupling.

Conclusions

The work presented in this brief report is an example for a very successful interdisciplinary cooperation, spanning the fields of physics, algorithmic design and programming techniques. The overall goal was to equip solid state physicists with a valuable tool to perform fast and reliable many-body calculations on modern supercomputers. Modifying an existing JD algorithm implementation by incorporating a new matrix data structure yielded excellent performance results, especially on vector processors. The implementation of the optimized program package on the supercomputer resources at HLRS provied new and exciting insights to the interaction of electronic and phononic degrees of freedom in the intensely studied MX-chains compounds.

Acknowledgements

We are particulary indebted to the C&C Research Laboratories, NEC Europe Ltd., for placing their JADA program package at our disposal. Special thanks go to the NIC Jülich, SSC Karlsruhe, LRZ München and HLR Stuttgart for the generous granting of their parallel computer facilities. We acknowledge useful discussions with A.R. Bishop and H. Büttner.

References

1. Cullum, J. K., Willoughby, R. A., *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, volume I & II, Birkhäuser, (Boston 1985).
2. Davidson, E. R.: J. Comput. Phys., **17**, 87 (1975).
3. G. L. G. Sleipjen and H. A. van der Vorst, SIAM J. Matrix Anal. Appl., **17**, 401 (1996).
4. A. Basermann and B. Steffen, *New Preconditioned Solvers for Large Sparse Eigenvalue Problems on Massively Parallel Computers*, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, CD-ROM, SIAM, (Philadelphia 1997).
5. R. Barrett at al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia (1993)

6. N. Kosugi, J. Comput. Phys., **55**, 426 (1984).
7. A. Basermann and B. Steffen, *Preconditioned solvers for large eigenvalue problems on massively parallel computers and workstation clusters*, in Parallel Computing: Fundamentals, Applications and New Directions, eds. E.H. D'Hollander, G.R. Joubert, F.J. Peters, and U. Trottenberg, Elsevier Science B. V., 1998, pp. 565–572.
8. A. Basermann, *Parallel preconditioned solvers for large sparse Hermitian eigenvalue problems*, Jack Dongarra and Vicente Hernandez, eds.: VECPAR'98 - Third International Conference for Vector and Parallel Processing, Lecture Notes in Computer Science, Springer, Berlin, 1573 (1999), pp. 72–85.
9. Bäuml, B., Wellein, G., Fehske, H,: Phys. Rev. B **58**, 3663 (1998).
10. G. Wellein and H. Fehske,*Towards the Limits of present-day Supercomputers: Exact Diagonalization of Strongly Correlated Electron-Phonon Systems*, E. Krause and W. Jäger, eds.: High Performance Computing in Science and Engineering 99, Springer, Berlin (1999), pp. 112–129.
11. B. I. Swanson et al., Phys. Rev. Lett. **82**, 3288 (1999).
12. S. P. Love et al., Phys. Rev. B **47**, 11107 (1993).