Analyzing Quantum Systems Using the Cray MTA-2

Wendell Anderson, *Naval Research Laboratory,* **Marco Lanzagorta**, *Scientific & Engineering Solutions and* **C. Stephen Hellberg**, *Naval Research Laboratory*

ABSTRACT: *Analysis of the low-temperature thermodynamic properties of many-body quantum systems requires the determination of the smallest eigenvalues of a very sparse matrix. Techniques developed at NRL to solve this problem use a Lanczos method that requires many multiplications of the sparse matrix by a dense vector, an operation ideally suited to the mutli-threaded and large uniformly accessible memory architecture of the Cray MTA-2.*

1. Introduction

The Naval Research Laboratory has recently installed and is operating a 40-processor Cray Multi-Threaded Architecture (MTA-2) computer. The MTA-2 with its hardware support for multi-threading and large uniformly accessible memory offers a promising new paradigm in parallel computing, especially for large problems.

One of the first scientific problems transitioned to the NRL MTA-2 was the study of the behavior of the charged ordered phase of manganites. The analysis of the properties of these materials involves the determination of the lowest eigenvalues of hundreds of square sparse matrixes, some with dimensions greater than ten million. Sufficient resources have not been available in order to perform all of the computations required to completely understand the problem.

Algorithms for determining the eigenvalues of sparse matrices are limited by the rate at which data may be retrieved from memory and not by the number of floating point operations that can be done per second. These algorithms suffer from delays associated with retrieving data from memory seriously degrading performance on a conventional machine. Many authors have looked at methods to reorder the calculations in order to reduce the delays associated with retrieving data from memory. Several authors [1-3] used register blocking and data pre-fetching to speed up the calculations. Others have examined the data structures of the sparse matrices to improve cache re-use [4-6]. There is even a toolbox (Sparsity) [7,8] that allow users to automatically generate kernels that are tuned to their matrices

Many of these considerations are not needed on the MTA-2 as the time to access data from any part of the memory is the same. That this type of problem is well-suited to the MTA has been noted [9-10].

2. Problem

The application considered here is associated with manganites. The manganites are a class of material with a general chemical formula of $(R,A)_{n+1}Mn_nO_{3n+1}$, the so-called Ruddelsden-Popper series [11] . Here R is a trivalent rare-earth ion and A is a divalent alkaline-earth ion. As $n \rightarrow \infty$ the compounds become $(R,A)MnO_3$ with a three-dimensional perovskite structure. For n=1, the compound $(R,A)_2MnO_4$ is a single-layer perovskite and for n=2, the compound $(R,A)_4Mn_2O_7$ is a bi-layer perovskite.

The various manganites exhibit a huge variety of phases including ferromagnetism, anti-ferromagnetism, and charge-ordered phases. Various theoretical models have been used to explain different aspects of their phase diagrams.

The current work focuses on the charge-ordered (CO) phase. Electron doping is controlled with the ratio of trivalent to divalent ions, $R_{1-x}A_x$. It is simplest to think about this phase at x=1/2, where there are equal amounts of Mn^{3+} and Mn^{4+}. The different charge states order in real space in a checkerboard pattern. The oxygens relax away from the Mn^{3+} ions and towards the Mn^{4+} ions, thus providing a repulsive potential between Mn^{3+} ions (or equivalently between Mn^{4+} ions). When this phase is favored, the potential energy gain of forming this pattern exceeds the kinetic energy loss due to the formation of the insulating state.

The CO phase is generally the lowest temperature phase, but the CO phase has been seen to melt with decreasing temperature in some manganites, including the three dimensional $Pr_{0.65}(Ca_{0.7}Sr_{0..30})_{0.35}MnO_3$ [12] and bi-layer $LaSr_2Mn_2O_7$ [13]. In these cases, the lowest temperature phase is metallic, and the CO insulator is only observed at intermediate temperatures. Theoretically, a re-

entrant transition has been obtained using extended Hubbard models both with electron-phonon interactions [14] and without electron-phonon interactions [15].

Re-entrant behavior is never seen in single-layer manganites. Here the charge-order transition in the extended Hubbard model (without electron-phonon interactions) on a finite periodic two-dimensional square lattice is studied in order to investigate why single-layer manganites never show re-entrant behavior. Previous work [15] solved this model in infinite spatial dimensions, resulting in finite entropy (due to the spins) at T=0 in the CO phase, so a re-entrant transition was guaranteed to be found. This model has also been studied on a 16-site periodic cluster that is effectively four-dimensional due to interactions around the periodic boundaries [16,17]. In this case, a re-entrant transition was also found.

In a system with dimensionality greater than two, the spin ordering in the CO phase is frustrated. The anti-ferromagnetic interactions are next-nearest neighbor on a cubic lattice. In the infinite two-dimensional square lattice, the next-nearest-neighbor anti-ferromagnetic interaction is not frustrated and the spins will order into an unfrustrated Néel state with zero entropy at T=0. Thus it may be possible to avoid the re-entrant transition.

The extended Hubbard Hamiltonian is given by

$$H = t\sum_{<ij>\sigma}\left(c_{i\sigma}^{+}c_{j\sigma} + h.c.\right) + U\sum_{i} n_{i\uparrow}n_{i\downarrow} + V\sum_{<ij>} n_i n_j \qquad (1)$$

where $c_{i\sigma}^{+}$ ($c_{i\sigma}$) creates (annihilates) an electron with spin σ on site i, $n_{i\sigma}$ is the number operator with spin σ on site i and $n_i = n_{i\uparrow} + n_{i\downarrow}$. The hopping amplitude is t, $<ij>$ enumerates nearest neighbor sites on the two-dimensional square lattice, U is on-site repulsion, and V is the nearest-neighbor repulsion. Good introductions to the Hubbard model can be found in Ref. [18-19]. The non-interacting bandwidth on the two-dimensional square lattice is given by W=8|t|; U is set equal to W and V is varied at quarter filling (one electron for every two sites). For small V, the ground state is expected to be a homogeneous Fermi liquid. For large V, the electrons will crystallize in a checkerboard pattern to avoid occupying neighboring sites.

The Hamiltonian (1) is solved on a 20-site two-dimensional periodic cluster using a recently developed finite-temperature Lanczos technique [16,20,21]. This cluster is the smallest possible two-dimensional cluster allowing no multiple next-nearest-neighbor interactions around the boundary. Periodic boundary conditions are chosen resulting in a closed Fermionic shell in the non-interacting limit [22]. From the eigenvalues (E_m) of the Hamiltonian, the expectation value of the energy as a function of temperature can be calculated from

$$E = \left(\sum_m E_m \exp(-\beta E_m)\right) \Big/ \sum_m \exp(-\beta E_m) \qquad (2)$$

where $\beta = 1/k_B T$ and k_B is Boltzmann's constant, As $T \to 0$ temperature, $E \to E_0$ the lowest energy eigenvalue. At low temperatures, the sums in (2) can be approximated by the sums over the smallest eigenvalues. From E the susceptibility χ is given by

$$\chi = \frac{1}{N}\frac{\partial E}{\partial V} \qquad (3)$$

where N=20 is the number of sites calculated. In order to calculate the susceptibility χ the Hamiltonian must be calculated over a range of V's. Changing the value of V affects only the diagonal elements of the matrix and the Hamiltonian matrix M can be written as

$$M = S + V \cdot D; \qquad V = 0.2, ..., 6.0 \qquad (4)$$

where S is a sparse matrix, and D is a diagonal matrix. The susceptibility χ is equivalent to the nearest-neighbor pair correlation function by the Kubo formula. In the homogeneous phase χ is large, but in the CO phase where there is only a small probability of finding electrons on neighboring sites χ is small.

3. Lanczos Algorithm

The Lanczos algorithm is most commonly used to calculate the lowest (or highest) eigenvalue of a matrix [13]. The algorithm starts with a normalized random vector, and generates a series of orthogonal vectors by multiplying the current vector by the Hamiltonian **M** and orthogonalizing:

$$v_{n+1} = Mv_n - a_n v_n - b_n v_{n-1} \qquad (5)$$

where

$$a_n = \frac{v_n \cdot M \cdot v_n}{v_n \cdot v_n} \quad , \quad b_n^2 = \frac{v_n \cdot v_n}{v_{n-1} \cdot v_{n-1}} \qquad (6)$$

It is easy to show that $v_m \cdot v_n = 0$ if $n \neq m$. In the new basis, the Hamiltonian matrix has tri-diagonal form,

$$M' = \begin{pmatrix} a_0 & b_1 & 0 & 0\cdots \\ b_1 & a_1 & b_2 & 0\cdots \\ 0 & b_2 & a_2 & b_3\cdots \\ 0 & 0 & b_3 & a_3\cdots \\ \vdots & \vdots & \vdots & \vdots\ddots \end{pmatrix} \qquad (7)$$

M' can then be diagonalized by conventional means.

Because most of the calculations of the Lanczos coefficients occur in the matrix-vector multiplication, the algorithm is particularly efficient with large sparse matrices, such as Hubbard Hamiltonians. In theory, the Lanczos algorithm transforms an entire matrix into tri-diagonal form, but in practice there are some subtleties.

In principle, one could use the Lanczos algorithm to find all eigenvalues of a matrix [14]. Once M' is the same size as M, their eigenvalues are identical, assuming the computations are done with infinite precision. In practice, however, the algorithm is very susceptible to round-off errors. After a given eigenvalue has converged, the subsequent Lanczos vectors should be orthogonal to it, but are not due to the finite precision of the computation. The rounding errors are amplified by the Lanczos algorithm, and generate spurious eigenvalues. These eventually converge to become multiple copies of previously determined eigenvalues. One might think that enforcing orthogonality is the only way to solve this problem. This is however a very expensive proposition if many eigenvalues are needed. The most efficient way to determine the lowest (and highest) eigenvalues is simply to identify and discard the spurious eigenvalues at the end of the Lanczos procedure.

4. Computational problem

At half filling, there will be 10 electrons on the 20 sites of the cluster. The ground state will have 5 electrons with up spin and 5 electrons with down spin, but we have to allow for spin flips in the excited states. The total number of ways to place 10 electrons with arbitrary spin on 20-sites is

$$N_H = \sum_i \binom{20}{i}\binom{20}{10-i} = 847{,}660{,}528 \quad (8)$$

ways and the resultant Hamiltonian matrix has dimensions N_H x N_H. Using both translational and point-group symmetries, the matrix M can be written as a block diagonal matrix

$$M = \begin{pmatrix} M_1 & 0 & \cdots & 0 \\ 0 & M_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & M_{156} \end{pmatrix} \quad (9)$$

The eigenvalues of M can be found by calculating the eigenvalues of the M_k matrices. The largest irreducible M_k block is complex Hermitian and has dimension 12,018,700. The M_k matrixes are very sparse with only about 33 non-zero elements per column and row on average. Some of the M_k are identical and thus the problem can be reduced to calculating the eigenvalues of the 66 distinct M_k's and then accounting for the eigenvalue multiplicity induced by the multiple copies. The matrix may be re-ordered so that the first 66 are the distinct sub-matrices and the rest are copies of one of these. Performing the calculations over the range of values of V requires that the smallest eigenvalues of 1980 matrices be determined.

For each M_k, we perform N_L=4000 Lanczos steps. Spurious eigenvalues are removed [23], leaving more than 2000 real eigenvalues in each sector (although some of these will not yet have converged). The extreme eigenvalues (lowest and highest) converge first [21].

The computationally demanding part of the Lanczos algorithm is the matrix-vector multiplication, that is performed N_L times for each of the 1980 matrices. The remaining parts of the algorithm are vector-vector operations and vector-scalar multiplication.

To calculate the Lanczos coefficients from (5) and (6), $v_{-1} = 0; v_0 = r/|r|$ where r is a vector of random numbers dawn from a uniform distribution. Then

$$\begin{aligned} Y &= M_k v_i \\ a_i &= v_i Y \\ Z &= Y - a_i v_i + v_{i-1} \\ b_i &= \sqrt{Z \cdot Z} \\ v_{i+1} &= Z / b_i \\ v_i &= -b_i v_i \end{aligned} \quad (10)$$

is recursively calculated for i=0,1,…,N_L-1.

The number of hardware operations (where an operation is a multiply, add, or multiply/add pair) and memory accesses required for the calculation of each (a_i,b_{i+1}) pair can be obtained by summing the number of operations/references required by each step of the recursion and are given in the table below

Type	Operations	Memory Accesses
Real	$9N_{M_k} + Q_{M_k}$	$14N_{M_k} + 3Q_{M_k}$
Complex	$16N_{M_k} + 4Q_{M_k}$	$24N_{M_k} + 5Q_{M_k}$

where N_{M_k} is the size of the M_k matrix and Q_{M_k} is the number of non-zero off-diagonal elements of M_k plus the 2 N_{M_k} elements need to define the diagonal of M_k (4).

The same number of operations and memory references are required for every value of V. For a specific M_k, the total number of operations to calculate the smallest eigenvalues for one value of V is

$$N_L * \left(9 \sum_{M_k real} N_{M_k} + \sum_{M_k real} Q_{M_k} + 16 \sum_{M_k imag} N_{M_k} + 4 \sum_{M_k imag} Q_{M_k} \right) \tag{11}$$

and the number of memory references is

$$N_L * \left(14 \sum_{M_k real} N_{M_k} + 3 \sum_{M_k real} Q_{S_k} + 24 \sum_{M_k imag} N_{M_k} + 5 \sum_{M_k imag} Q_{M_k} \right) \tag{12}$$

For this application we have

$$\begin{aligned} \sum_{M_k real} N_{M_k} &= 54,401,940 \\ \sum_{M_k real} Q_{M_k} &= 1,792,156,768 \\ \sum_{M_k imag} N_{M_k} &= 122,403,588 \\ \sum_{M_k imag} Q_{M_k} &= 4,032,937,872 \end{aligned} \tag{13}$$

using equations (10) and (11) and recalling $N_L = 4000$ and $N_V = 30$, then determining all of the Lanczos coefficients will require 2.4e+15 hardware operations and 3.5e+15 memory accesses. .On a single processor that could access 64-bit data at 100 MHZ sustained this part of the problem would require more than 400 days.

5. Previous Implementation

The application was originally implemented to run on multiple processors as a C++ program using the Message Passing Interface (MPI) standard under the auspices of the High Performance Computer Modernization Program Office's (HPCMPO) Common High Performance Computing Software Support Initiative (CHSSI). The program has been set-up to generate the elements of one of the irreducible matrices with each processor responsible for generating the coefficients for a set of rows of the matrix, The program then calculates the Lanczos coefficients a_i and b_i by having each processor perform the part of the sparse matrix dense vector calculation for the rows of the matrix stored in that processor. This requires that each processor have the entire dense vector and that all processes are updated at each step with the new values of the v_{i+1} vectors. When all of the a_i and b_i coefficients have been calculated for the matrix, they are saved to a disk file and then transferred to a PC where the smallest eigenvalues of the Heisenberg matrix are determined. The code has been ported to the IBM SP2 and SP3, the SGI Origin 2000 and 3000, and the COMPAQ ES45 and GS320.

This implementation has two aspects that limit scalability. First, each processor holds the entire vector, so no matter how many processors are used the total amount of memory required on each processor increases as the dimension of the matrix. Second, at each Lanczos step, the portion of the vector updated by a processor must be communicated to every other processor. The amount of data that must be sent to other processors is proportional to the number of processors used in the calculation. Even with these restrictions, this implementation has been used to solve for the lowest eigenvalues of complex matrices with dimension as large as 70,000,000 [14].

6. MTA Implementation

Two factors drove the decisions on how to use the MTA to solve the problem. The first was the desire to use the uniform memory access feature and multiple threading paradigm of the MTA to reduce the time required to find the eigenvalues of the Hamiltonian Matrix. The second was to minimize the amount of recoding. Originally the thought was to simply convert the existing code to the MTA replacing the existing structure with a shared memory version. This would entail removing the MPI calls and placing explicit parallelization pragmas through out the program (as the default on the MTA for C/C++ is not to parallelize loops.)

Analysis of the problem indicated that almost all of the time (>95%) would be spent in the calculation of the Lanczos coefficients, an operation that was just a small part of the overall C++ code. As a result, the decision was made to use the current code to generate the matrices and add code to store the matrices on disk. A program would then be written for the MTA to read this file, calculate the Lanczos coefficients for a series of matrices based on different values of V and save the Lanczos coefficients in a disk file. Post processing then could be performed on a PC to determine the eigenvalues of interest. The code was written in Fortran.

The matrices are stored in a modified compressed row storage (CRS) format using three vectors R, I, and J. R contains the diagonal elements of the S and D matrixes and the nonzero off diagonal elements of the M' matrix as encountered in a row order fashion. Each element in the J vector contains the column number of the corresponding element in the R vector and each element in the I vector contains the location in the R vector of the start of the data

for that row with the last element of the I vector containing the size of he R vector. For example if

$$M = \begin{pmatrix} 2.0 & 1.1 & 3.1 \\ 1.1 & 0.0 & 0.0 \\ 3.1 & 0.0 & 0.0 \end{pmatrix}, \quad D = \begin{pmatrix} 1.8 & 0.0 & 0.0 \\ 0.0 & 2.5 & 0.0 \\ 0.0 & 0.0. & 0.0 \end{pmatrix} \quad (14)$$

then

R	2.0	1.8	1.1	3.1	0.0	2.5	1.1	0.0	0.0	3.1

J	1	1	2	3	2	2	1	3	3	1

I	O	4	7	10

The disk file begins with a header containing the number of Lanczos iterations, the dimension of the matrix, the weight to be given to the matrix, matrix type - real or complex, number of symmetries and an entry for each symmetry (describing its type). The header is followed by the I, J, and R vectors.

After a file has been produced, the data is copied to the scratch disks on the MTA. The Fortran program lanczos is then run to determine for the range of V's the Lanczos coefficients for the Hamiltonian Matrices. The coefficients for each V are stored in a separate file. If the program does not complete normally, the program is restartable, checking for the V's already completed and not recalculating the lanczos coefficients if they have already been stored to disk.

The code for the calculation of the lanczos coefficients is quite simple taking only 14 lines of code with the first 6 devoted to the sparse matrix vector multiple

```
Do index = 1, Imax
 Y(index) = 0
 Do index2 = I(index)+1, I(index+1)
  Y(index) = Y(index)+R(index2)*Vvec(J(index2))
 end do
end do
aval(iteration) = DOT_PRODUCT(Y,Vvec)
Uvec=Y+aval(iteration)*Vvec+Uvec
bval(iteration)=sqrt(DOT_PRODUCT(Uvec,Uvec)
Do index = 1, Imax
 Tmp = Vvec(index)
 Vvec(index)=Uvec(index)/bval(iteration)
 Uvec(index)=-Tmp*bval(iteration)
end do
```

6. Timing results

Given the large number of operations/memory references required for this problem it was not possible to time the entire problem Instead the times to generate 1000 pairs of Lanczos coefficients for one V for the largest M_k (12,018,700 x 12,018,700 complex Hermitian matrix) were examined. Timings were run on the IBM P3 and COMPAQ ES45 at the MSHPC at the Aeronautical Systems Center (ASC), the SGI Origin 3800 at ERDC (US Army Engineer Research and Development Center) and MTA at NRL (Naval Research Laboratory). Running times for 1000 iterations of the Lanczos code kernel (i.e. no I/O times are included) are given in the table below.

Platform	Speed MHz	16P mines	32P mines	Speed Up
IBM P3	375	142.9	111.6	1.28
COMPAQ ES45	1000	90.1	63.5	1.42
SGI Origin	400	158.4	160.6	0.99
MTA	200	28.2	16.6	1.73

The most obvious result is the comparison of the total time required to calculate 1000 pairs of lanczos coefficients from M. On 32 processors the MTA was four times faster than the COMPAQ ES45, seven times faster than the IBM P3, and almost ten times faster than the Origin 3800, even though the MTA has the slowest clock. The speedup from 16 processors to 32 processors was also best on the MTA with an improvement of 75%.

Since this application is constrained by the speed of retrieving data from memory, the layout of the data is critical. On a conventional machine, the goal is to have data as close to the processor as possible. Thus the MPI program divided the matrix into groups of rows and assigned one group to each processor. Each processor then calculates the output vector for its group of rows. On the other hand, on the MTA, data is physically hashed so that even if a user is running on one processor, his data is distributed across all of the processors. This technique eliminates worst-case memory access patterns, so that contention is minimized. It also allows simple method for a program getting access to all of the memory in the system, even if it is running on a single processor. The table below shows for 100 Lanczos iterations for the large complex matrix how the MTA scales with the number of processors.

Processors	Time (secs)	Speedup	Linear
4	633.5		
8	322.4	1.96	2.00
12	214.9	2.95	3.00
16	164.6	3.85	4.00
20	135.7	4.68	5.00
24	117.1	5.41	6.00
28	108.0	5.86	7.00
32	99.4	6.37	8.00
36	94.9	6.67	9.00
40	91.6	6.92	10.00

Scaling proceeds fairly well until we get up to about 24 processors. Then the scaling rate of increase begins to falls off. The reason for this is that in its current configuration the machine cannot achieve the aggregate 8GHZ bandwidth of the 40 200MHZ processors issuing a 64bit read request every cycle. The best rates that have been seen at NRL are about 2.7 GHZ. Currently, Cray is currently designing and building a top plane for the NRL machine that should allow the machine to actually achieve over 95% of the maximum bandwidth.

If one wanted to find the Lanczos coefficients for two different V's (matrices F_1 and F_2), the user has two options on an Origin with 32 processors. The first option would be to use all 32 processors to calculate the coefficients for F_1 and when this was completed use the 32 processors to calculate the coefficients for F_2. Total wall-clock time would be 320 minutes. The other option would be to divide the 32-processor system into two 16-processor sub-hypercubes and to run F_1 on one sub-hypercube and F_2 on the other Since there would be no memory contention, wall clock time would be only 158 minutes. A similar scheme could be used on the IBM and COMPAQ's. However, splitting the MTA into two sets of processors and running F_1 and F_1 at the same time would not result in comparable speed-ups as the two programs would compete for the memory bandwidth.

6. Conclusions

The MTA proved to be a very good resource for this application. By analyses of the computational requirements of the problem, the amount of recoding required was kept to a minimum. The actual code written was small, required no MTA specific constructs, and is general enough to be easily used to solve other problems. In the current MTA configuration, the program ran 4 times faster on 32 processors, than on 32 processors of any of the other HPC machines that had been previously used. Further improvements are expected when the MTA top plane is installed this summer.

Acknowledgments

The authors are deeply indebted to the Cray staff in Seattle especially Preston Briggs, Jace Mogill, and Greg Woodman for their support and willingness to answer questions. The MPI code was developed by Steve Hellberg under the High Performance Computer Modernization Program Office's (HPCMPO) Common High Performance Computing Software Support Initiative (CHSSI). Computations were performed on resources provided by the DOD HPCMPO at the Aeronautical Systems Center (ASC), US Army Engineer Research and Development Center (ERDC), and Naval Research Laboratory (NRL) shared resource centers.

About the Authors

Wendell Anderson is a mathematician and the head of the Research Computers Section of the Center of Computational Science at the Naval Research Laboratory. He can be reached at Naval Research Laboratory, Code 5593, 4555 Overlook Avenue Washington, D. C. 20375 or E-mail wendell.anderson@nrl.navy.mil. C. Stephen Hellberg is a research physicist in the Center for Computational Materials Science at the Naval Research Lab, , Code 6390, 4555 Overlook Avenue Washington, D. C. 20375. His interests include strongly correlated materials and quantum computing. Marco Lanzagorta is a physicist for Scientific & Engineering Solutions. His interests include High Performance Computing and Visualization. He can be reached at Naval Research Laboratory, Code 5593, 4555 Overlook Avenue Washington, D. C. 20375,

References

1 S. Toledo," Improving the memory-system performance of sparse Matrix vector multiplication", IBM Journal Research Development, Vol. 41 No. 6 Nov 97.

2 R. Geus and S. Rollin, "Towards a Fast Parallel Sparse Matrix-Vector Multiplication" at Parallel Computing conference (PARCO 99). Delft, The Netherlands Aug, 1999

3 Ali Pinarand Michael T. Heath**, "**Improving Performance of Sparse Matrix-Vector Multiplication" Supercomputing 1999, Portland, Oregon, November , 1999

4 W. Gropp, D. Kaushik, Keyes, and B. Smith, "Improving the Performance of Sparse Matrix-vector Multiplication by Blocking ", SIAM Annual Meeting July 2000

5 Leonid Oliker, Xiaoye Li, Parry Husbands, Rupak Biswas "Effects of Ordering Strategies and Programming Paradigms on Sparse Matrix Computations", SIAM Review Volume 44, Number 3 pp. 373-393

6 R. S. Tuminaro, J. N. Shadid, S. A. Hutchinson Parallel Sparse Matrix Vector Multiply Software for Matrices with Data Locality *Concurrency: Practice and Experience*, 10(3), pp. 229-247, March 1998.

7 E. Im and K. A. Yelick "Optimizing Sparse Matrix-Vector Multiplication for Register Reuse ",International Conference on Computational Science, San Francisco, California, May 2001

8 E. Im and K. A. Yelick, "Optimizing Sparse Matrix Vector Multiplication on SMP's " SIAM Conf. Parallel Processing for Scientific Computing, San Antonio, TX, March 1999.

9. Allan Snavely and Larry Carter (SDSC/UCSD) with Jay Boisseau and Amit Majumdar (SDSC), Kang Su Gatlin and Nick Mitchell (UCSD), John Feo and Brian Koblenz

(Tera Computer) Multi-processor Performance on the Tera MTA , Supercomputing98 (Orlando, Fl. Nov 1998)

10. L. Carter, J. Feo, and A. Snavely , “Performance and Programming Experience on the Tera MTA”, Supercomputing SIAM conference on Parallel Processing for Scientific Computing (March 1999)

11. M. B. Salamon and M. Jaime, "The physics of manganites: Structure and transport", Rev. Mod. Phys. **73**, 583 (2001).

12 Y. Tomioka, A. Asamitsu, H. Kuwahara, and Y. Tokura, “Reentrant transition of the charge-ordered state in perovskite manganites”’ J.. Phys. Soc. Japan **66**, 302 (1997).

13 T. Kimura, R. Kumai, Y. Tokura, J. Q. Li, and Y. Matsui, "Successive structural transitions coupled with magnetotransport properties in $LaSr_2Mn_2O_7$ "Phys. Rev. B **58**, 11081 (1998).

14 Q. Yuan and P. Thalmeier, "Reentrant charge ordering caused by polaron formation", Phys. Rev. Lett. **83**,3502 (1999).

15 R. Pietig, R. Bulla, and S. Blawid, "Reentrant charge order transition in the extended Hubbard model" Phys. Rev. Lett. **82**, 4046 (1999).

16 C. S. Hellberg, "Theory of the reentrant charge-order transition in the manganites",J. Appl. Phys. **89**, 6627 (2001).

17 E. Dagotto, R. Joynt, A. Moreo, S. Bacci, and E. Gagliano, "Strongly correlated electronic systems with one hole: dynamical properties", Phys. Rev. B **41**, 9049 (1990).

18 A. Auerbach, *Interacting Electrons and Quantum Magnetism* (Springer, New York, 1994).

19 P. Fazekas, *Lecture notes on electron correlation and magnetism* (World Scientific, Singapore, 1999),

20 C. S. Hellberg, W. E. Pickett, L. L. Boyer, H. T. Stokes, and M. J. Mehl, "Abs initial calculation of spin gap behavior in CaV_4O_9 , " J. Phys. Soc. Japan **68**, 3489 (1999).

21. C. S. Hellberg, "Low-Temperature Thermodynamics of Quantum Systems" in *Computer Simulation Studies in Condensed Matter Physics XIII*, edited by D. P. Landau, S. P. Lewis, and H. B. Schüttler (Springer-Verlag, Heidelberg, Berlin, 2000).

22 C. S. Hellberg and E. Manousakis, “Green's-function Monte Carlo for lattice fermions: Application to the t-J model” Phys. Rev. B **61**, 11787 (2000).

23 J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations* (Birkhauser, Boston, 1985).

24 C. S. Hellberg and S. C. Erwin, "Strongly correlated electrons on a silicon surface: theory of a Mott insulator" Phys. Rev. Lett. **83**, 1003 (1999).