

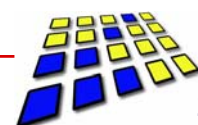
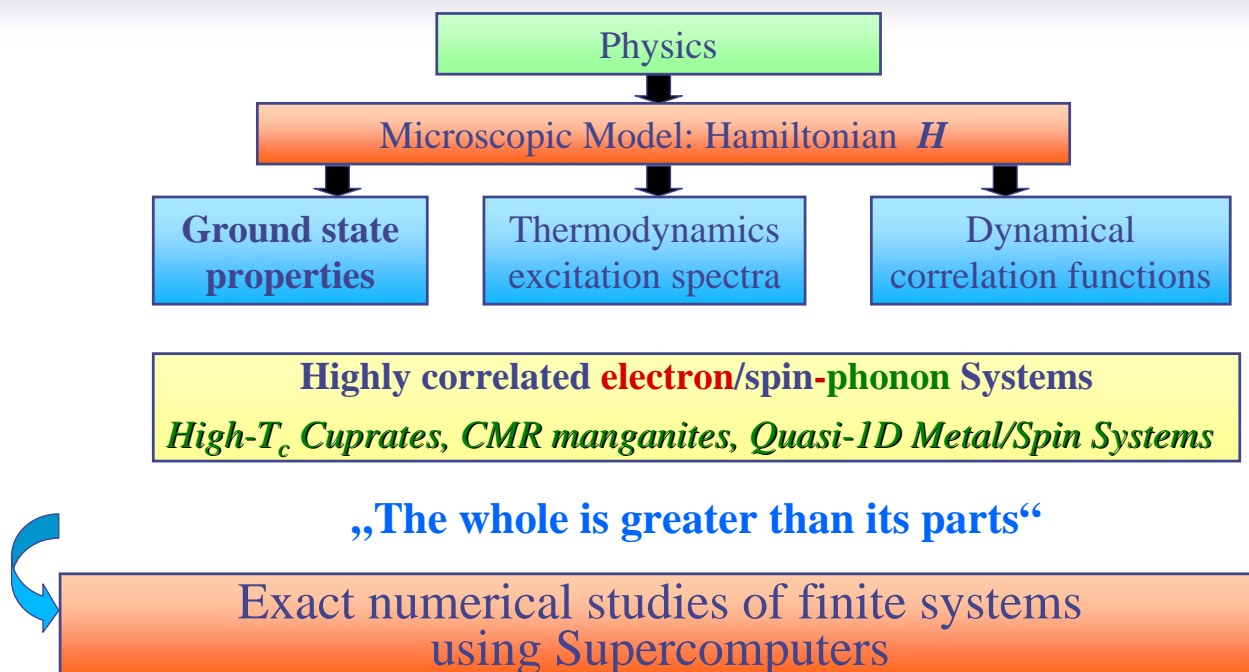
Parallelization Strategies for Density Matrix Renormalization Group algorithms on Shared-Memory Systems

G. Hager HPC Services, Computing Center Erlangen, Germany
E. Jeckelmann Theoretical Physics, Univ. Mainz, Germany
H. Fehske Theoretical Physics, Univ. Greifswald, Germany
G. Wellein HPC Services, Computing Center Erlangen, Germany



1

Motivation – From Physics to Supercomputers

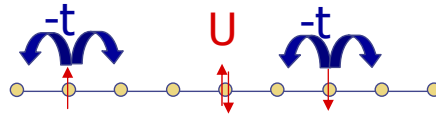


2

Motivation – Microscopic Models

- Microscopic Hamiltonians in second quantization
e.g. Hubbard model

$$H = -t \sum_{\langle ij \rangle, \sigma} [c_{i\sigma}^\dagger c_{j\sigma} + \text{H.c.}] + U \sum_i n_{i\uparrow} n_{i\downarrow}$$



e.g. Holstein-Hubbard model (HHM)

$$H = -t \sum_{\langle ij \rangle, \sigma} [c_{i\sigma}^\dagger c_{j\sigma} + \text{H.c.}] + U \sum_i n_{i\uparrow} n_{i\downarrow} + g\omega_0 \sum_{i, \sigma} (b_i^\dagger + b_i) n_{i\sigma} + \omega_0 \sum_i b_i^\dagger b_i$$

HHM: Coupling between electrons and phonons (lattice oscillations)

- Hilbert space / #quantum states growth exponentially

HHM using an N-site lattice: $4^N * (M+1)^N$ ($N \sim 10-100$; $M \sim 10$)

Electrons Phonons: Max. M per Site



3

Motivation – Numerical Approaches

Traditional Approaches

- Quantum Monte Carlo (QMC)
 - Exact Diagonalization (ED)
- Massively Parallel Codes on Supercomputers

New Approach

- Density Matrix Renormalization Group (DMRG) Method
 - Originally introduced by White in 1992
 - Large sequential C++ package is in wide use (quantum physics and quantum chemistry)
 - Elapsed Times: hours to weeks
 - No parallel implementation available to date

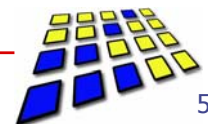
Aim: Parallelization strategy for DMRG package



4

Algorithms




- Summary: Exact Diagonalization
- DRMG algorithm

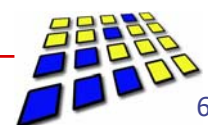


5

Algorithms – Exact Diagonalization

Exact Diagonalization (ED)

- Chose **COMPLETE** basis set (e.g. localized states in real space):  Sparse matrix representation of **H**
- Exploit conservation laws and symmetries to reduce to effective Hilbert space by a factor of $\sim N$ (still exponential growth in N)
- Perform **ONE** Exact Diagonalization step using iterative algorithms e.g. Lanczos or Davidson
- Sparse Matrix-Vector Multiply determines computational effort  Do not store matrix!
- ED on TFlop/s computers:
N=8 ; M=7  Matrix Dimension ~ 10 billion
- **No approximations !!**



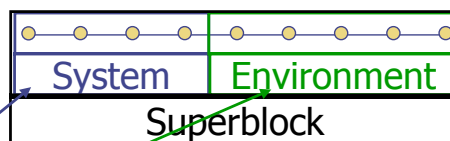
6

Algorithms - DMRG

- Basic Idea: Find an appropriate (reduced) basis set describing the ground-state of H with high accuracy

- Basic Quantities:

- Superblock = system & environment



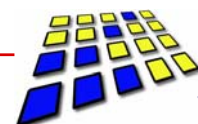
- Superblock state (product of system & environment states)

$$|\psi\rangle = \sum_{ij} \psi_{ij} |i\rangle |j\rangle$$

- Reduced density matrix (DM) (summation over environment states)

$$\rho_{ii'} = \sum_j \psi_{ij}^* \psi_{i'j}$$

- Eigenstates of DM with largest eigenvalues have most impact on observables !!!



7

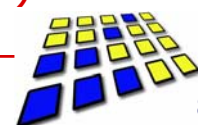
Algorithm - DMRG

DMRG algorithm (finite size; left to right sweep)

1. Diagonalize the reduced DM for a **system block** of size l and extract the **m eigenvectors with largest eigenvalue**
2. Construct all relevant operators (**system block** & **environment**,...) for a **system block** of size $l+1$ in the reduced density matrix eigenbasis
3. Form a superblock Hamiltonian from **system** & **environment** Hamiltonians plus two single sites
4. Diagonalize the new superblock Hamiltonian



Accuracy depends mainly on m ($m \sim 100 - 10000$)



8

Algorithm - DMRG

Implementation

- Start-Up with infinite-size algorithm
- DM diagonalization: LAPACK (dsyev) costs about 5 %
- Superblock diagonalization costs about 90 % (Davidson algorithm)
- Most time-consuming step: Sparse matrix-vector multiply (MVM) in Davidson (costs about 85 %)
- Sparse matrix H is constructed by the transformations of each operator in H :

$$H_{ij;i'j'} = \sum_{\alpha} A_{ii'}^{\alpha} B_{jj'}^{\alpha}$$

Contribution from **system block** and from **environment**



9

Algorithm - DMRG

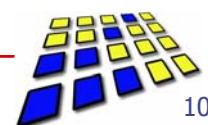
Implementation of sparse MVM (1)

- Sparse MVM: Sum over dense matrix-matrix multiplies!

$$\sum_{i'j'} H_{ij;i'j'} \psi_{i'j'} = \sum_{\alpha} \sum_{i'} A_{ii'}^{\alpha} \sum_{j'} B_{jj'}^{\alpha} \psi_{i'j'}$$

- However A and B may contain only a few nonzero elements, e.g. if conservation laws (quantum numbers) have to be obeyed
- To minimize overhead an additional loop (running over nonzero blocks only) is introduced

$$\begin{aligned} H\psi &= \sum_{\alpha} \sum_k (H\psi)_{L(k)}^{\alpha} \\ &= \sum_{\alpha} \sum_k A_k^{\alpha} \psi_{R(k)} [B^T]_k^{\alpha} \end{aligned}$$



10

DMRG

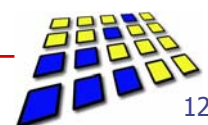
- Benchmark Systems
- Benchmark Cases
- Single Processor Performance
- Potential Parallelization approaches
- Parallel BLAS
- OpenMP Parallelization



11

DMRG: Benchmark Systems

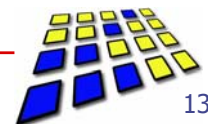
Processor	Frequency	Peak Performance	System	Memory arch.	#Processor Size / Max
IBM Power4	1.3 GHz	5.2 GFlop/s	IBM p690	ccNUMA	32 / 32
Intel Xeon DP	2.4 GHz	4.8 GFlop/s	Intel860	UMA	2 / 2
Intel Itanium2	1.0 GHz	4.0 GFlop/s	HP rx5670	UMA	4 / 4
UltraSparc III	0.9 GHz	1.8 GFlop/s	SunFire 6800	ccNUMA	24 / 24
MIPS R14000	0.5 GHz	1.0 GFlop/s	SGI O3400 SGI O3800	ccNUMA	28 / 32 128 / 512



12

DMRG: Benchmark Cases

- Case1: 2D - 4X4 periodic Hubbard model at half filling
 - $U=4, t_{x/y}=1$
 - Small number of lattice sites (16)
 - large values of m ($m \sim 1000 - 10000$) are required to achieve convergence in 2D
- Case2: 1D - 8 site periodic Holstein-Hubbard model at half filling
 - $U=3, t=1, \omega_0=1, g^2=2$
 - Max. 6 phonons per site ->Phonons are implemented as pseudo-sites -> large effective lattice size (~ 50)
 - m is at most 1000
- Metrics:
 - $P(N)$ is total performance on N CPUs
 - Speed-Up: $S(N) = P(N) / P(1)$
 - Parallel efficiency: $\varepsilon(N) = S(N) / N$

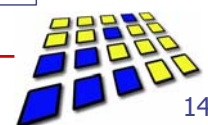
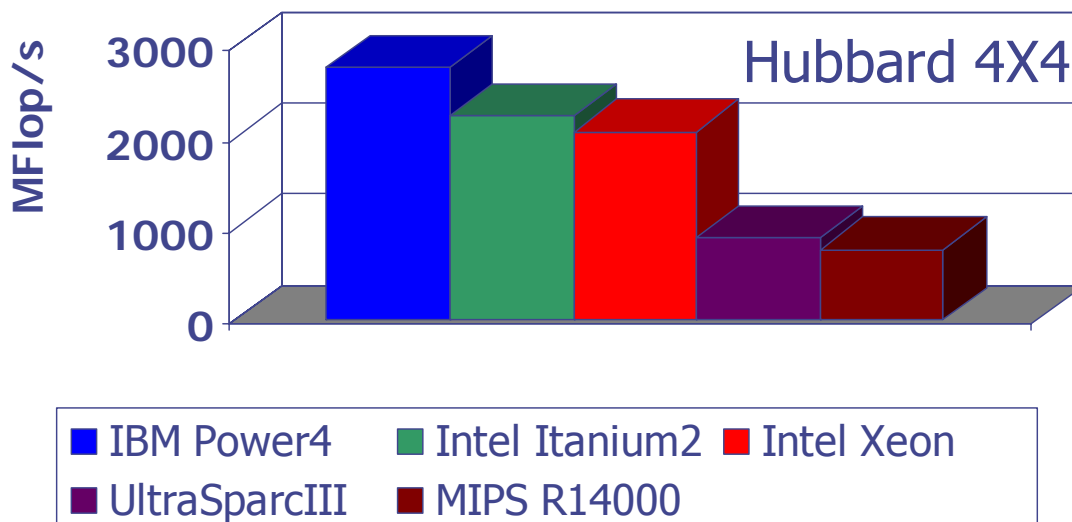


DMRG: Single Processor Performance

Numerical core: DGEMM



High sustained single processor performance

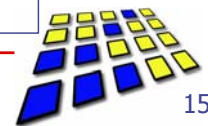
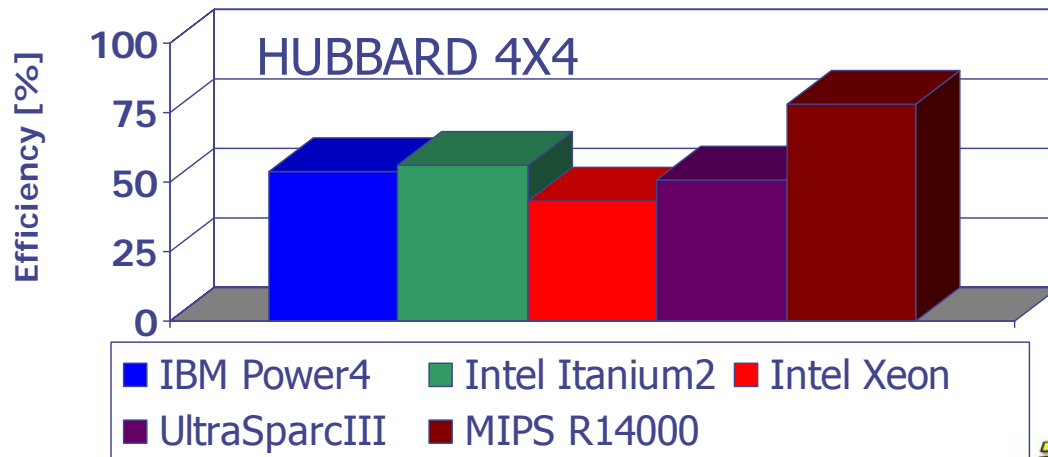


DMRG: Single Processor Efficiency

Processor Efficiency: Fraction of Peak Performance achieved

Parameters:

- DGEMM implementation (proprietary)
- C++ compiler (proprietary)



15

DMRG: Potential Parallelization approaches

Implementation of sparse MVM - pseudocode

$$H\psi = \sum_{\alpha} \sum_k A_k^{\alpha} \psi_{R(k)} [B^T]_k^{\alpha}$$

// W: wavevector ; R: result

for (α=0; α < number_of_hamiltonian_terms; α++) { ← Parallel loop !?

term = hamiltonian_terms[α];

for (k=0 ; k < term.number_of_blocks; k++) ← Parallel loop !?

li = term[k].left_index;
ri = term[k].right_index;

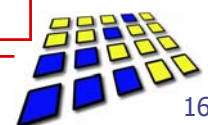
temp_matrix = term[k].B.transpose() * W[ri];

R[li] += term[k].A * temp_matrix;

}}

Data dependency !

Matrix-Matrix-Multiply
(Parallel DGEMM ?!)



16

DMRG: Potential Parallelization approaches

Parallelization strategies

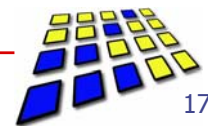
1. Linking with parallel BLAS (DGEMM)
 - Does not require restructuring of code
 - Significant speed-up only for large (transformation) matrices (A, B)
2. Shared-Memory parallelization of outer loops
 - Chose `openMP` for portability reasons
 - Requires some restructuring & directives
 - Speed-Up should not depend on size of (transformation) matrices

Expected maximum speed-up for total program:

- if MVM is parallelized only: $\sim 6 - 8$
- if also Davidson algorithm is parallelized: ~ 10

MPI parallelization

- Requires complete restructuring of algorithm -> Ian?

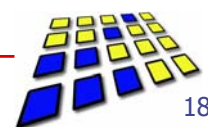
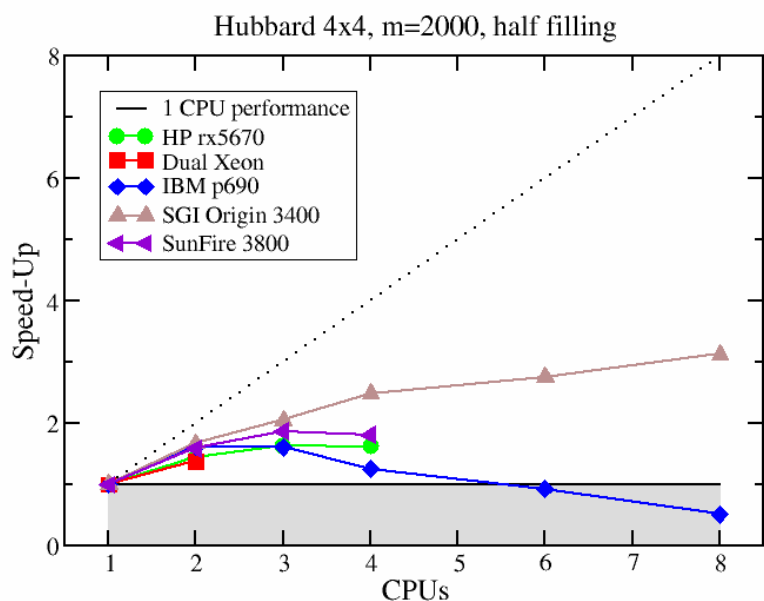


17

DMRG: Parallel BLAS

Linking with parallel BLAS

- Useless on IBM for #CPU > 4
- Best scalability on SGI (Network, BLAS implementation)
- Dual processor nodes can reduce elapsed runtime by about 30 %
- Increasing m to 7000: $S(4) = 3.2$
- Small m (~ 600) with HHM: No Speed-Up

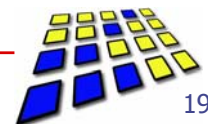


18

DMRG: OpenMP Parallelization

OpenMP Parallelization

- Parallelization of innermost **k** loop: Scales badly
 - loop too short
 - collective thread operations within outer loop
- Parallelization of outer **α** loop: Scales badly
 - even shorter
 - load imbalance (trip count of **k** loop depends on **α**)
- Solution:
 - "Fuse" both loops (**α** & **k**)
 - Protect write operation **R[li]** with **lock** mechanism
 - Use list of OpenMP **locks** for each block **li**



19

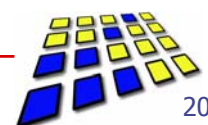
DMRG: OpenMP Parallelization

Implementation of parallel sparse MVM – pseudocode (prologue loops)

```
// store all block references in block_array
ics=0;
for ( $\alpha=0$ ;  $\alpha <$  number_of_hamiltonian_terms;  $\alpha++$ ) {
    term = hamiltonian_terms[ $\alpha$ ];
    for (k=0 ; k < term.number_of_blocks; k++) {
        block_array[ics]=&term[q];
        ics++;
    }
}
icsmax=ics;

// set up lock lists
for(i=0; i < MAX_NUMBER_OF_THREADS; i++)
    mm[i] = new Matrix // temp.matrix

for (i=0; I < MAX_NUMBER_OF_LOCKS; i++) {
    locks[i]= new omp_lock_t;
    omp_init_lock(locks[i]);
}
```



20

DMRG: OpenMP Parallelization

Implementation of parallel sparse MVM – pseudocode (main loop)

```

// W: wavevector ; R: result
#pragma omp parallel private(mymat, li, ri, myid, ics)
{
    myid = omp_get_thread_num();
    mytmat = mm[myid]; // temp thread local matrix

#pragma omp for
    for (ics=0; ics< icsmax; ics++) {

        li = block_array[ics]->left_index;
        ri = block_array[ics]->right_index;

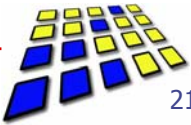
        mytmat = block_array[ics]->B.transpose() * W[ri];

        omp_set_lock(locks[li]);
        R[li] += block_array[ics]->A * mytmat;
        omp_unset_lock(locks[li]);
    }
}

```

Fused (α, k) loop

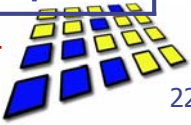
Protect each block of result vector **R** with locks



DMRG: OpenMP Parallelization

- The parallel code is compliant to the OpenMP standard
- However: **NO** system did compile and produce correct results with the initial MVM implementation!

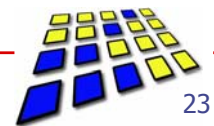
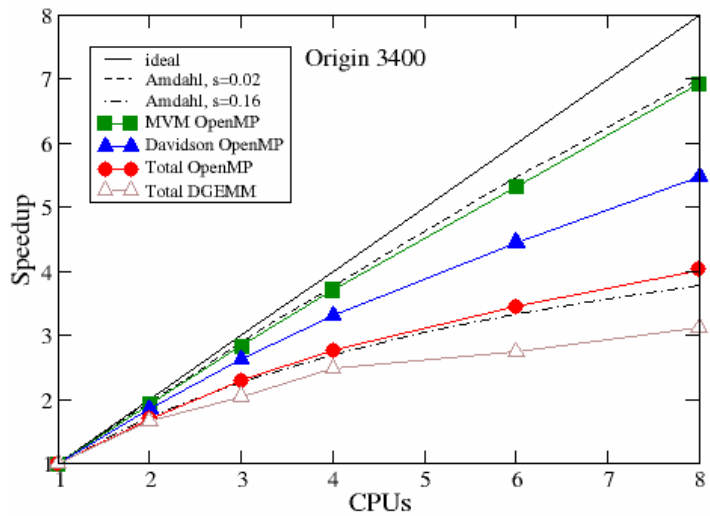
IBM xIC V6.0	OpenMP locks prevent <code>omp for</code> parallelization	Fixed by IBM
Intel efc V7 ifc V7	<code>omp for</code> Loop is not distributed (IA64) or produces garbage (IA32) → call to Intel	Does not work
SUN forte7	Does not allow <code>omp critical</code> inside C++ classes!	Does not work
SGI MIPSpro 7.3.1.3m	Complex data structures can not be allocated inside <code>omp parallel</code> regions	Allocate everything outside loop



DMRG : OpenMP Parallelization

Scalability on SGI Origin

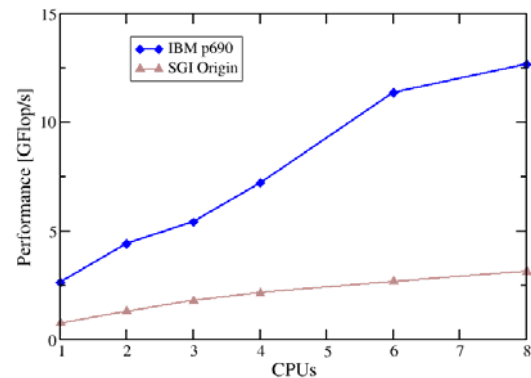
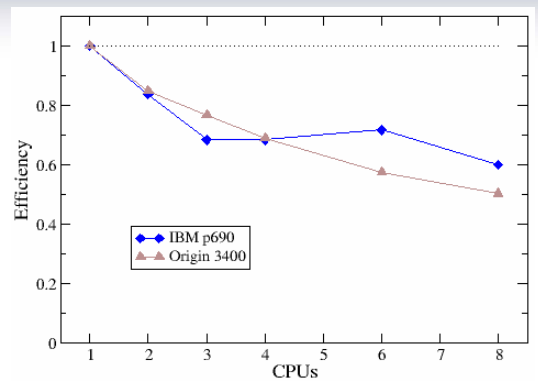
- `OMP_SCHEDULE=STATIC`
- OpenMP scales significantly better than parallel DGEMM
- Serial overhead in parallel MVM is only about 5%
- Parallelization of Davidson should be the next step



DMRG : OpenMP Parallelization

Scalability & Performance: SGI Origin vs. IBM p690

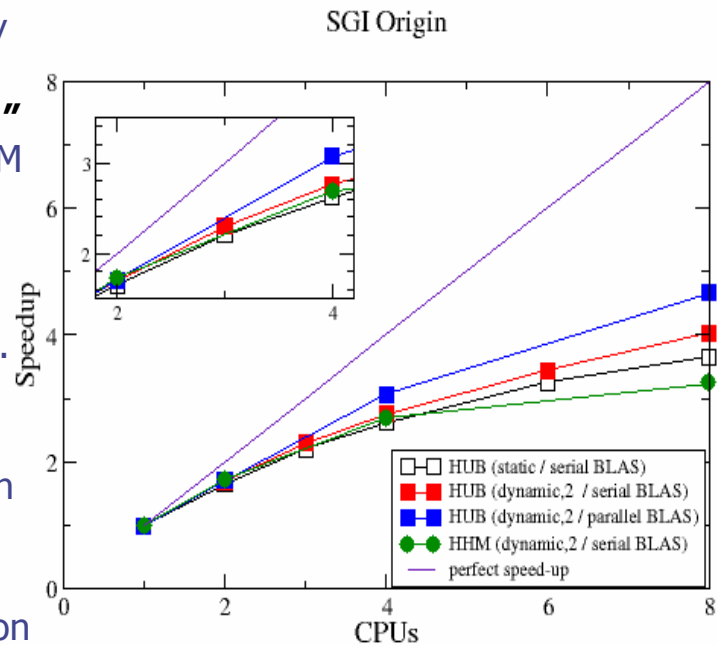
- Scalability is pretty much the same on both systems
- Single processor run and `OMP_NUM_THREADS=1` differ by approx. 5% on IBM
 - Hardly any difference in SGI
- Total performance
 - 1 * Power4 = 8 * MIPS
 - 8 * Power4 > 12 GFlop/s sustained !



DMRG : OpenMP Parallelization

Further improvement of total performance/scalability

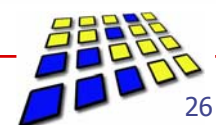
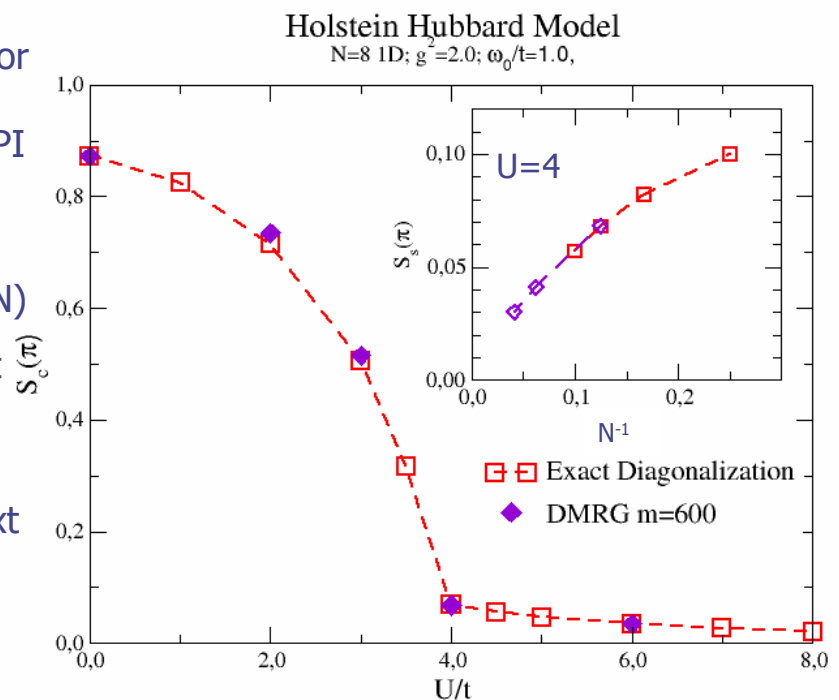
- Chose best distribution strategy for parallel for loop:
`OMP_SCHEDULE="dynamic,2"`
 (reduces serial overhead in MVM to 2%)
- Re-Link with parallel LAPACK /BLAS to speed-up density-matrix diagonalization (DSYEV). Good thing to do:
`OMP_NESTED=FALSE`
- HHM spends much more time in serial SuperBlock generation than Hubbard case
- ToDo: Parallelization of Davidson & SuperBlock generation!



25

Application: Peierls insulator (PI) – Mott insulator (MI) transition in the Holstein Hubbard Model

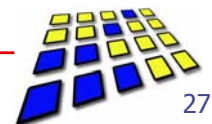
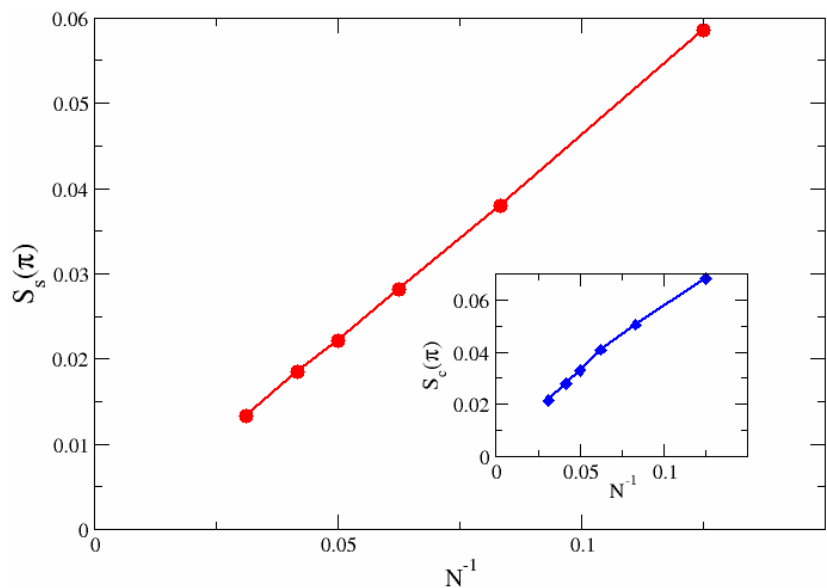
- Use charge-structure factor $S_c(\pi)$ to determine transition point between PI and MI
- $S_c(\pi)$ may depend significantly on lattice size (N)
- Exact Diagonalization: At most N=10 sites
- DMRG allows to study finite size effects (see next slide)
- At small lattice sizes ED and DMRG are in good agreement!



26

Application: Finite-Size Study of Spin & Charge Structure Factors in the half-filled 1D periodic HHM

- Parameters: $U=4$, $t=1$, $\omega_0=1$, $g^2=2$ (QC point – see prev. slide)
- 5 boson pseudosites
- DMRG can get to very large lattices (up to 32 sites)
- Strong support for the conjecture that $S_c(\pi)$ and $S_s(\pi)$ vanish at quantum critical point in the thermodynamic limit ($N \rightarrow \infty$)

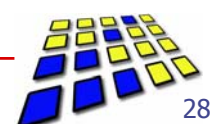


27

Application: Peierls insulator (PI) – Mott insulator (MI) transition in the Holstein Hubbard Model

DMRG computational requirements compared to ED

ED ($N=8$; Mat. Dim $\sim 10^{10}$)	1024 CPUs (Hitachi SR8k)	~ 12 Hrs. elapsed time	600 GB
DMRG ($N=8$; $m=600$)	1 CPU (SGI Origin)	~ 18 Hrs. elapsed time	2 GB
DMRG ($N=24$; $m=1000$)	4 CPU (SGI Origin)	~ 72 Hrs. elapsed time	10 GB



28

Summary

Existing DMRG code from quantum physics/chemistry:

- Kernel: sparse Matrix-Vector-Multiply (MVM)
- Approaches for shared-memory parallelization of MVM (parallel BLAS vs. OpenMP)
- Fusing inner & outer loop allows a scalable OpenMP implementation for MVM routine with a parallel efficiency of 98% for MVM
- May compute ground-state properties for 1D Holstein-Hubbard model at high accuracy with minimal computational requirements (when compared to ED)
- SMP parallelization has still some optimization potential, but more than 8 CPUs will presumably never be reasonable



Acknowledgement



Our work is funded by the Bavarian Network for High Performance Computing (KONWIHR)

