Shared-memory Parallel DMRG and its Application to Stripe Formation in Doped 6-Leg Hubbard Ladders

G. Hager[1], E. Jeckelmann[2], H. Fehske[3] and G. Wellein[1]

[1]Regionales Rechenzentrum Erlangen, [2]Johannes Gutenberg-Universität Mainz, [3]Ernst-Moritz-Arndt-Universität Greifswald

Abstract

We present two different approaches by which parallelization of the standard DMRG algorithm can be accomplished. The parallelized code shows good scalability for standard benchmark cases (2-dimensional periodic Hubbard model) up to at least eight processors and allows us to solve problems which exceed the capability of sequential DMRG calculations.
As an important application we investigate stripe formation in 6-leg cylindrical Hubbard ladders which are doped with holes away from half-filling. The parallel approach allows to consider systems with up to 28×6 sites at $m \approx 6000$ to 8000 on contemporary SMP systems.

Serial DMRG

Profiling analysis of the serial algorithm with a benchmark case (half-filled two-dimensional [4×4] Hubbard model at $U = 4$ with up to $m = 2000$) showed that 85 % of computing time is used in the sparse matrix-vector multiplication step of the Davidson algorithm that diagonalizes the superblock hamiltonian.
Even though the MVM is sparse, the dominating operation is dense matrix-matrix multiplication (MMM):

$$\sum_{i'j'} H_{ij;i'j'}\psi_{i'j'} = \sum_{\alpha}\sum_{i'} A^{\alpha}_{ii'}\sum_{j'} B^{\alpha}_{jj'}\psi_{i'j'} \qquad (1)$$

Considering the blocked structure of the components due to conservation laws and transistion rules and omitting matrix indices,

$$H\psi = \sum_{\alpha}\sum_{k} (H\psi)^{\alpha}_{L(k)} = \sum_{\alpha}\sum_{k} A^{\alpha}_{k}\psi_{R(k)}\left[B^{\mathrm{T}}\right]^{\alpha}_{k} \quad . \qquad (2)$$

Any of the four sums is a candidate for parallelization.

Approach 1: Parallel DGEMM

The easiest approach to parallel DMRG computation consists in using a shared-memory parallel dense matrix-matrix multiplication algorithm for the two inner sums (i' and j') in (1). This is available out of the box for all contemporary computer architectures in vendor-supplied libraries and can be optimized on RISC processors to yield a large fraction of peak performance [1].

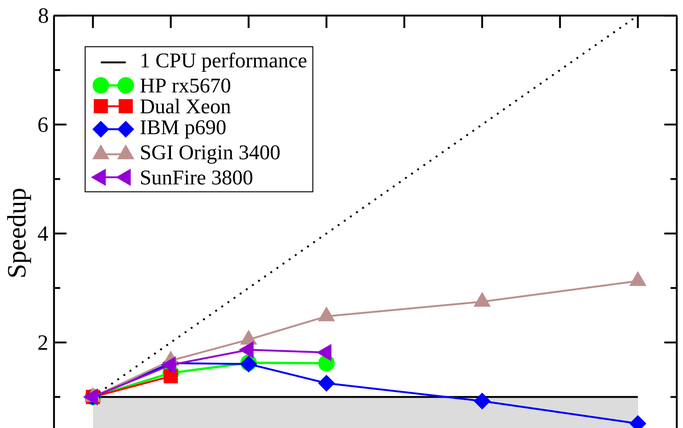

Fig. 1: *Speedup of DGEMM-parallel DMRG for the benchmark case*

Scalability for this case is poor, mainly because of overhead in the parallel computation with small matrices. The approach makes sense only for certain limiting cases where individual matrices are very large on the average. Additionally, the quality of the parallel DGEMM implementation plays a major role here.

Approach 2: Parallel sparse MVM

In order to get improved parallel efficiency, one can parallelize the reduction operation that makes up the outer sums in (2).
Parallelizing only one sum is inefficient because of load imbalance and large OpenMP loop overhead (short loops). Thus the outer loop nest must be transformed to a single loop. Original code:

```
// W is wave vector, R ist result
for(i=0; i < number_of_hamiltonian_terms; i++)
  {
    term = hamiltonian_terms[i];
    for(q=0; q < term.number_of_blocks; q++)
      {
        li = term[q].left_index;
        ri = term[q].right_index;

        temp_matrix = term[q].B.transpose() * W[ri];
        R[li] += term[q].A * temp_matrix;
      }
  }
```

Problems to note:

- The reduction operation on the result vector blocks `R[li]` introduces a race condition that must be avoided by serializing access to each `R[li]`.
- The inner loop length depends on the outer loop counter (term counter).

Transformed core:

```
#pragma omp parallel private(mytmat,li,ri,myid,ics)
{
  myid = omp_get_thread_num();
  mytmat = mm[myid]; // thread-local
#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.transp() * W[ri];

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

From profiling data, parallel speedups of up to 7 can be expected in the ideal case where parallelization is bare of overhead.

Scalability

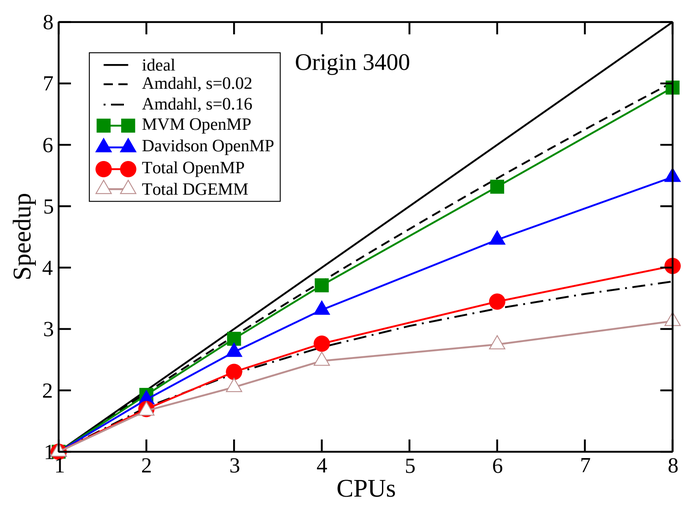

Fig. 2: *Speedup of MVM-parallel DMRG for the benchmark case (SGI Origin system)*

Scalability of the parallel MVM alone is very good (approx. 2 % serial fraction). As expected, by Amdahl's Law, performance is limited by the remaining serial portions. This can be somewhat improved by linking to a parallel BLAS library (+10–20 %).
Further improvements are possible:

- Lock-free version of MVM with completely private target data and reduction operations at the end (done)
- Identification of other parallelizable loops in the code (work in progress)

Application: Stripe formation

Whether stripe formation in Hubbard systems,

$$H_{\mathrm{HM}} = -t\sum_{\langle ij\rangle,\sigma}\left[c^{\dagger}_{i\sigma}c_{j\sigma} + \mathrm{H.c.}\right] + U\sum_{i} n_{i\uparrow}n_{i\downarrow} \quad , \qquad (3)$$

which are doped with holes away from half-filling and are subject to cylindrical (y-periodic and x-open) BCs is merely an artifact is an ongoing discussion [3]. We have performed parallel DMRG ground-state calculations with 14×6 (8 holes), 21×6 (12 holes) and 28×6 (16 holes) systems at $U = 12$ in order to clarify this issue. Results:

- With larger system size, transistion to "striped" state occurs at larger m.
- Using reflection symmetry to speed up the calculation is forbidden at 14×6 (non-convergence).

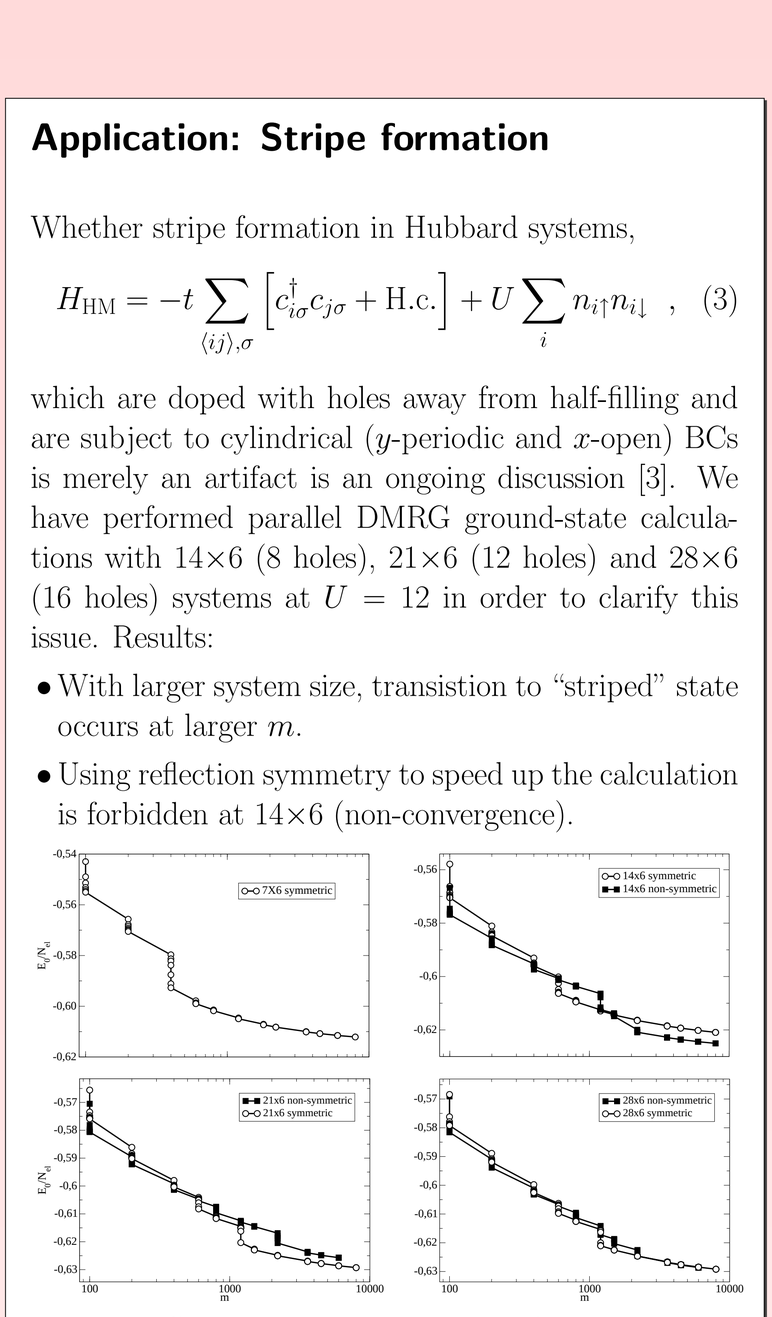

Fig. 3: *Ground-state energy vs. m for all systems considered*

- Clear stripe signatures for 14×6 and 21×6 systems (zero crossings of spin density).

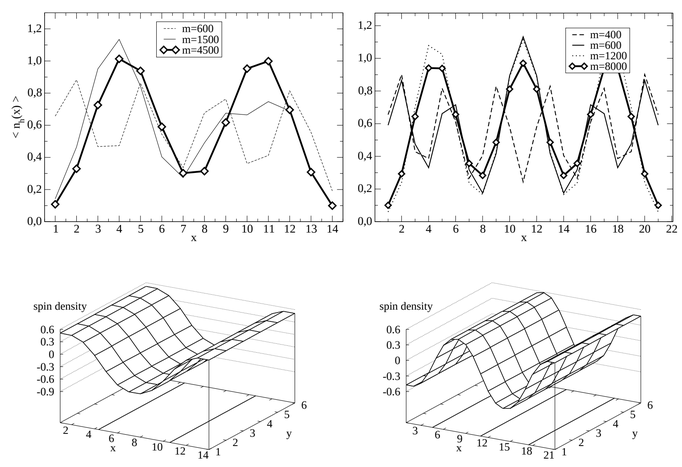

Fig. 4: *y-integrated hole density (above) and staggered spin density (below) for 14×6 and 21×6 systems*

- Results still inconclusive for 28×6 — no clear stripe signatures at $m = 6000$. No decision for symmetric vs. non-symmetric calculation possible.

The 28×6 runs required about four weeks of wallclock time and up to 100 GBytes of memory each on 8 CPUs of an IBM p690 node.

References

[1] G. Hager, F. Brechtefeld, P. Lammers and G. Wellein: Processor Architecture and Application Performance in Modern Supercomputers. InSiDE 1, No. 1, Spring 2003, pp. 8–13 (2003)

[2] G. Hager, E. Jeckelmann, H. Fehske and G. Wellein: Parallelization Strategies for Density Matrix Renormalization Group Algorithms on Shared-Memory Systems. J. Comp. Phys. **194**(2), 795 (2004)

[3] S. R. White and D. J. Scalapino: Stripes on a 6-leg Hubbard Ladder. Phys. Rev. Lett. **91**, 136403 (2003).