High-performance Parallel Diagonalization for Huge Matrices in Quantum Lattice-fermion Problems

S. Yamada (CCSE, Japan Atomic Energy Agency)
T. Imamura (The University of Electro-Communications, Japan)
T. Kano (CCSE, Japan Atomic Energy Agency)
M. Machida (CCSE, Japan Atomic Energy Agency)

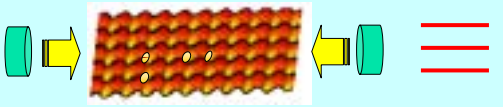
acknowledgments

Physics:
H. Matsumoto(Tohoku Univ., Japan)
Y. Ohashi (Keio Univ. ,Japan)
The Earth Simulator:
Staff members of the Earth Simulator center

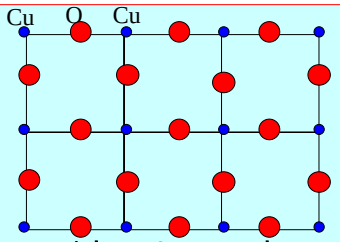
Outline

- Introduction
  - **Computational approaches for Hubbard model**
- Parallel computing for exact diagonalization
  - **Alternative parallel scheme for matrix-vector multiplication**
  - **New algorithm for exact diagonalization**
  - **Comparison with conventional method**
  - **Numerical test**
- Conclusion

Quantum Lattice-fermion Problems

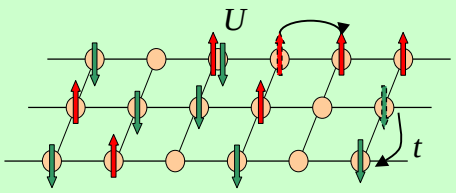


Periodical Potential (Optical Lattice)
In Atomic Fermi Gas



High-Tc Superconductor

The Simple Model: Hubbard Model



$$H_{Hubbard} = -t\sum_{i,j,\sigma}\left[a^{+}_{i,\sigma}a_{j,\sigma} + H.C.\right] + U\sum_{i} n_{i,\uparrow}n_{i,\downarrow}$$

Computational approach

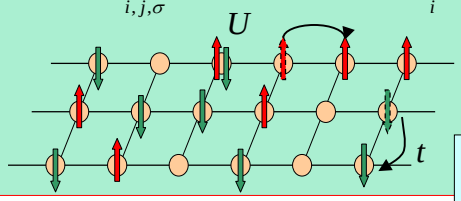
Computational approaches for the Hubbard model

approach	dimension of model	accuracy	memory usage
Quantum Monte Carlo	1D～3D	poor ～ good (negative sign)	small
DMRG (Density Matrix Renormalization Group)	1D	good	small
Exact Diagonalization (Solving the ground state)	1D ～3D	perfect	huge

"Diagonalization" is Exact, and it requires tough HPC techniques !

Hubbard Hamiltonian matrix

$$H_{Hubbard} = -t\sum_{i,j,\sigma}\left[a^{+}_{i,\sigma}a_{j,\sigma} + H.C.\right] + U\sum_{i} n_{i,\uparrow}n_{i,\downarrow}$$



N: No. of sites
n(↑): No. of up-spins
n(↓) : No. of down-spins

A state: $|0100101\rangle_{\uparrow}|0000100\rangle_{\downarrow}$ = A

$${}_{N}C_{n(\uparrow)} \times {}_{N}C_{n(\downarrow)}$$

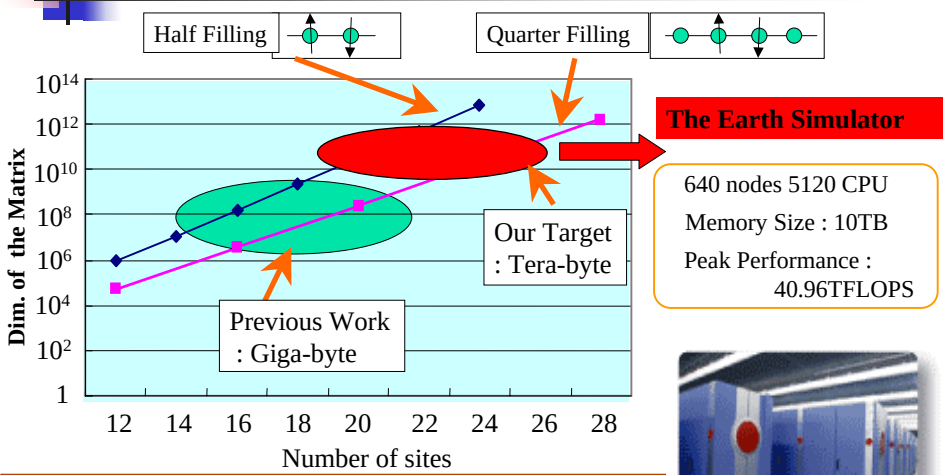
<A|H|A><A|H|B>

...

The Hamiltonian Matrix is Huge, Sparse, and Symmetric.

The "Lanczos" method has been traditionally employed.

Dimension of Hamiltonian matrix



The Earth Simulator

640 nodes 5120 CPU
Memory Size : 10TB
Peak Performance :
40.96TFLOPS

In "Exact diagonalization", the parallelization is essential to extend the system size.

Lanczos method

smallest eigenvalue and corresponding eigenvector
of huge, sparse and symmetric matrix

traditional way

Lanczos method

$x_0 \neq 0$

$\beta_0 = 0,\ v_{-1} = 0,\ v_0 = x_0 / \|x_0\|$

do k=0,m-1

$w = Hv_k - \beta_k v_{k-1}$

$\alpha_k = (w, v_k)$

$w = w - \alpha_k v_k$

$\beta_{k+1} = \sqrt{(w, w)}$

$v_{k+1} = w / \beta_{k+1}$

enddo

Algorithm of Lanczos method

$$\begin{pmatrix} \alpha_0 & \beta_1 & & & \\ \beta_1 & \alpha_1 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-2} & \alpha_{m-2} & \beta_{m-1} \\ & & & \beta_{m-1} & \alpha_{m-1} \end{pmatrix}$$

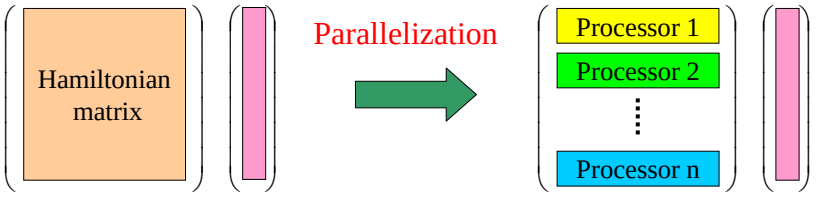
eigenvalue ,eigenvector

Most time consuming operation

Parallelization

Parallelization of Matrix-vector multiplication

Matrix-vector multiplication



Row-wise partitioning

The distribution of the non-zero elements of the matrix is irregular.

The load balancing of the calculation and communication is poor.

Alternative parallelization scheme

Matrix-vector product : Hv

$$H_{Hubbard} = I \otimes A + A \otimes I + D$$

⊗···direct product
A ··· sparse matrix
I ··· identity matrix
D ··· diagonal matrix

$$Hv_k = (I \otimes A)v_k + (A \otimes I)v_k + Dv_k$$

AV $\quad VA^T$ $\quad \overline{D} \bullet V$

$\bullet$ ··· element-wise product

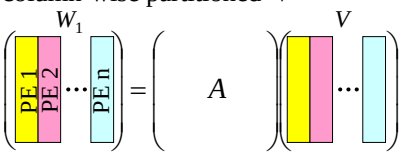
Transform

$$v = (v_1^T, v_2^T, \ldots, v_n^T)^T \rightarrow V = (v_1, v_2, \ldots, v_n)$$

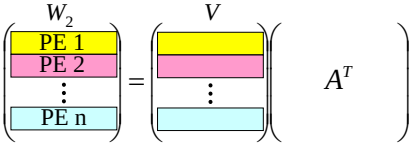
$$\mathrm{diag}(D) = (d_1^T, d_2^T, \ldots, d_n^T) \rightarrow \overline{D} = (d_1, d_2, \ldots, d_n)$$

Data distribution and communication

$W_1 = AV$
column-wise partitioned V

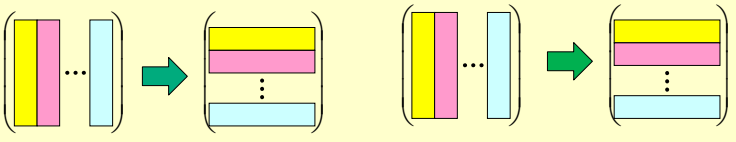


$W_2 = VA^T$
row-wise partitioned V



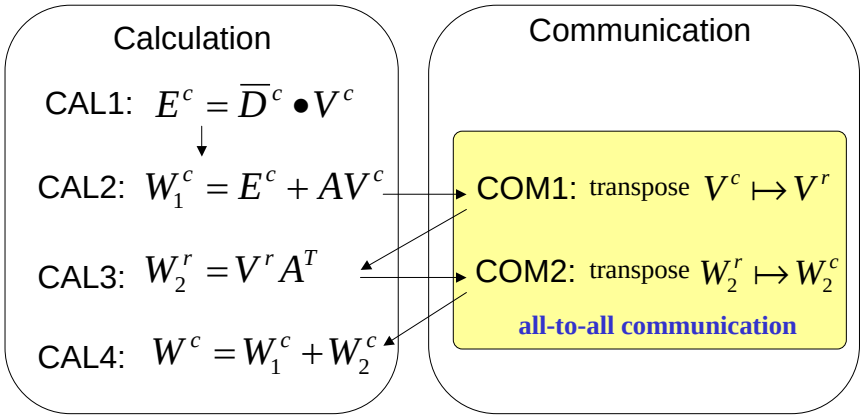
The calculation can be partitioned equally.

All-to-all communication is twice executed for the matrix transpose.



The amount of the communication data is equivalent for all processors.

Procedure of parallel computation



superscription

c : column-wise partitioning

r : row-wise partitioning

Lanczos method vs. CG method

memory usage	excellent (very small)
iteration control	**difficult** (fixed iteration count)
computation for eigenvector	**extra calculations**
convergence property	good
most consuming operation	matrix-vector multiplication

⇨ **Weak points** for HPC

⇩

CG method

Minimization for Reyleigh quotient

$$\mu(x) = \frac{x^T H x}{x^T x}$$

by CG method

CG method (LOBPCG)

Minimization for Reyleigh quotient $\mu(x) = \dfrac{x^T H x}{x^T x}$ by CG method

(Knyazev,1999)

$x_0 \neq 0$ (an initial guess), $p_0 = 0$
do k=0, ... until convergence
  $\mu_k = (x_k, Hx_k)/(x_k, x_k)$
  $w_k = T(Hx_k - \mu_k x_k)$
  $S_A = \{w_k, x_k, p_k\}^T H \{w_k, x_k, p_k\}$
  $S_B = \{w_k, x_k, p_k\}^T \{w_k, x_k, p_k\}$
  Solve the smallest eigenvalue μ
    and corresponding vector v,
    $S_A v = \mu S_B v$, $v = (\alpha, \beta, \gamma)^T$.
  $x_{k+1} = \alpha w_k + \beta x_k + \gamma p_k$
  $p_{k+1} = \alpha w_k + \gamma p_k$
enddo

three matrix-vector multiplications per iteration

x_0 := an initial guess, $p_0 := 0$
$x_0 := x_0/\|x_0\|$, $X_0 := Hx_0$, $P_0 = 0$, $\mu_{-1} := (x_0, X_0)$
$w_0 := X_0 - \mu_{-1} x_0$
do k=0, ... until convergence
  $W_k := Hw_k$
  $S_A := \{w_k, x_k, p_k\}^T \{W_k, X_k, P_k\}$
  $S_B := \{w_k, x_k, p_k\}^T \{w_k, x_k, p_k\}$
  Solve the smallest eigenvalue μ
    and the corresponding vector v,
    $S_A v = \mu S_B v$, $v = (\alpha, \beta, \gamma)^T$.
  $\mu_k := (\mu + (x_k, X_k))/2$
  $x_{k+1} := \alpha w_k + \beta x_k + \gamma p_k$, $x_{k+1} := x_{k+1}/\|x_{k+1}\|$
  $p_{k+1} := \alpha w_k + \gamma p_k$, $p_{k+1} := p_{k+1}/\|p_{k+1}\|$
  $X_{k+1} := \alpha W_k + \beta X_k + \gamma P_k$, $X_{k+1} := X_{k+1}/\|x_{k+1}\|$
  $P_{k+1} := \alpha W_k + \gamma P_k$, $P_{k+1} := P_{k+1}/\|p_{k+1}\|$
  $w_{k+1} := T(X_{k+1} - \mu_k x_{k+1})$, $w_{k+1} := w_{k+1}/\|w_{k+1}\|$
enddo

one matrix-vector multiplication per iteration

Preconditioner for CG method

Preconditioner : $T \approx H^{-1}$

- No preconditioning
- Point Jacobi (diagonal scaling, $T=D^{-1}$)
- Zero-shift Point Jacobi ($T=(D-\mu I)^{-1}$)
- Block Jacobi
- Neumann-polynomial expansion
- SSOR-type iteration

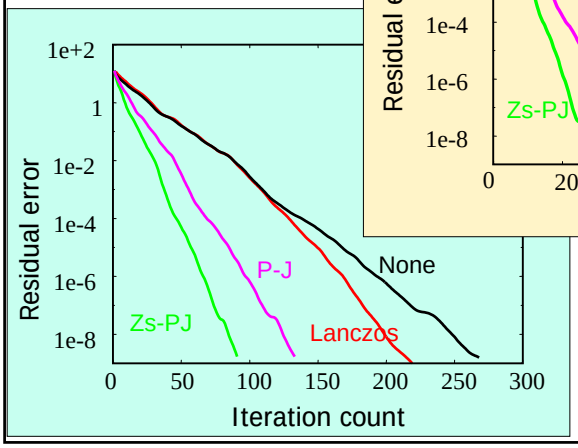
⇒ no data communication, no extra data storage

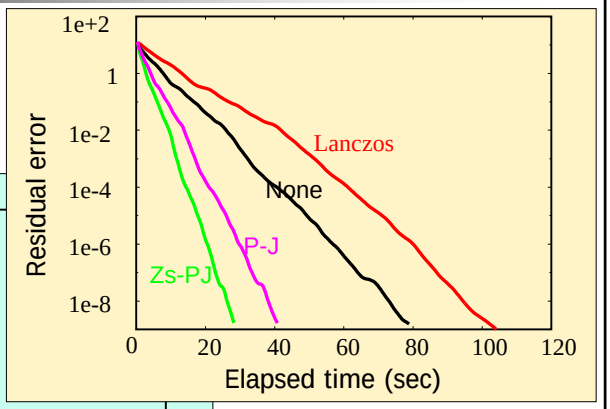
Problem: 1-D Hubbard model (1,502,337,600-dim.) 20-site 12 fermions (6↑,6↓)

Computation: 80 PE's of the Earth Simulator

Percontitioner	none	P-J	Zs-PJ	Lanczos
iterations	268	133	91	200 (fixed)
Time (sec)	78.9	40.8	28.2	95.0
GFLOPS	382.6	384.0	391.4	269.5

Preconditioner for CG method





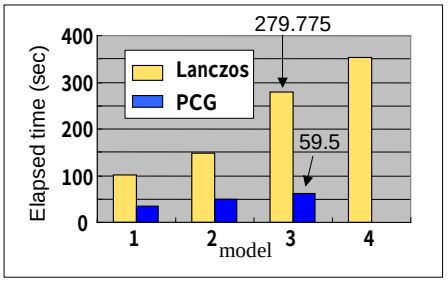
Lanczos method vs. CG method

	Lanczos method		(P)CG method
memory usage	excellent (very small)	>	1.5 times larger (on parallel computing)
iteration control	difficult (fixed iteration count)	<	**easy** (use of residual error)
computation for eigenvector	extra calculation	<	**no extra calculation**
convergence property	good	<	excellent (dependence on preconditioner)
matrix-vector multiplication	once per iteration	=	once per iteration

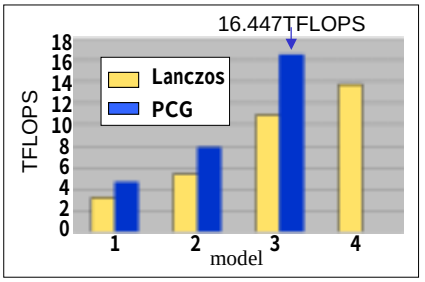
Numerical test

Dimension of Hamiltonian matrix

Model	No. of Sites	No. of Fermions		Dim. of H	No. of Nodes	Memory (TB)	
		↑-spin	↓-spin			Lanczos	PCG
1	24	6	6	18,116,083,216	128	0.8	1.3
2	21	8	8	41,408,180,100	256	1.9	2.9
3	22	8	8	102,252,852,900	512	4.6	6.9
4	22	9	8	**159,059,993,400**	512	7.1	(10.7)



Elapsed time



Performance

Conclusion

•We adopted the PCG method as the eigenvalue solvers for Hamiltonian matrices for the Hubbard model.

•We confirmed that the PCG method can solve the eigenvalue problems much faster than the Lanczos method.

•Maximal performance is 16.447TFLOPS (50%of the peak) by the PCG method

•We solved 159-billion-dimensional Hamiltonian matrix by the Lanczos method.