Parallelization of DQMC Simulations for Strongly Correlated Electron Systems

Che-Rung Lee

Dept. of Computer Science
National Tsing-Hua University
Taiwan

joint work with
I-Hsin Chung (IBM Research), Zhaojun Bai (UCDavis)

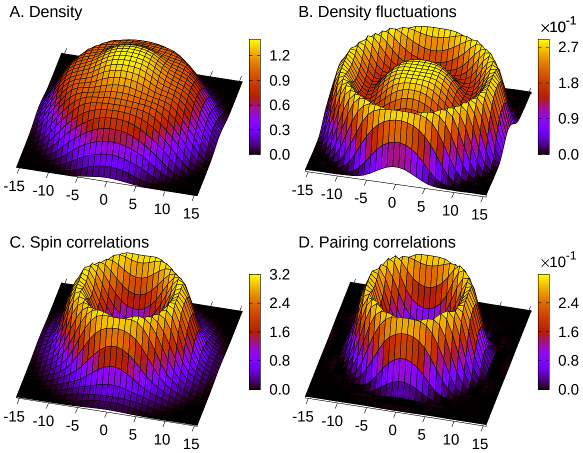
IEEE International Parallel and Distributed Processing Symposium 2010

Outline

1. DQMC simulations
2. DQMC parallelization
    - Algorithmic approaches
    - System approaches
3. Experiment results
4. Conclusion

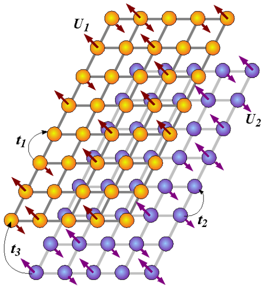
Computational Material Science

Understanding and exploiting the properties of solid-state materials: magnetism, metal-insulator transition, high temperature superconductivity, ...



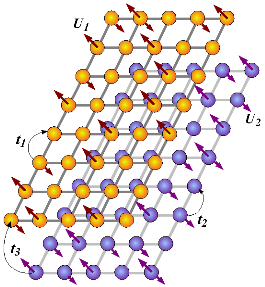
Hubbard Model and DQMC Simulations

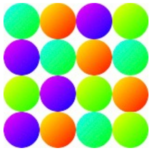
Many body simulation on multi-layer lattices using Hubbard model and quantum monte carlo method.



Hubbard Model and DQMC Simulations

Many body simulation on multi-layer lattices using Hubbard model and quantum monte carlo method.





QUEST (QUantum Electron Simulation Toolbox): Fortran 90 package for Determinant Quantum Monte Carlo (DQMC) simulations.

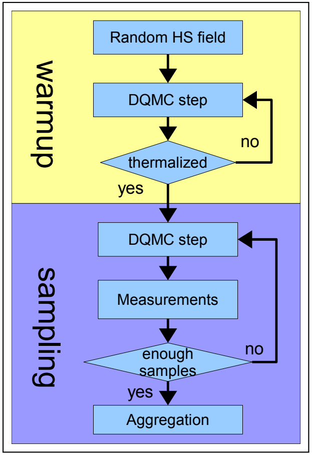
DQMC Algorithm

Two stages:

- Warmup stage
- Sampling stage

A DQMC step

1. Propose a local change: $h \to h'$.
2. Throw a random number $0 < r < 1$.
3. Accept the change if $r < \frac{\det(e^{-\beta H(h')})}{\det(e^{-\beta H(h)})}$.



Computational Kernels

The equal time Green's function

$$G_k = (I + B_k B_{k+1} \cdots B_1 B_L \cdots B_{k-1})^{-1}$$

Computational Kernels

The equal time Green's function

$$G_k = (I + B_k B_{k+1} \cdots B_1 B_L \cdots B_{k-1})^{-1}$$

The unequal time Green's function

$$G^\tau = \begin{pmatrix} I & & & B_1 \\ -B_2 & I & & \\ & \ddots & \ddots & \\ & & -B_L & I \end{pmatrix}^{-1}$$

Computational Kernels

The equal time Green's function

$$G_k = (I + B_k B_{k+1} \cdots B_1 B_L \cdots B_{k-1})^{-1}$$

The unequal time Green's function

$$G^\tau = \begin{pmatrix} I & & & B_1 \\ -B_2 & I & & \\ & \ddots & \ddots & \\ & & -B_L & I \end{pmatrix}^{-1}$$

Physical measurements

Operations on G_k and G^τ, Fourier Transform, etc.

Computational Challenges

- For simulating strongly correlated electron systems
  - The size of lattices need be large.
  - A longer warmup stage is required.

Computational Challenges

- For simulating strongly correlated electron systems
  - The size of lattices need be large.
  - A longer warmup stage is required.
- Numerical stability issues.
  - Additional stabilizing steps are required.
  - Most calculations need double precision.
  - Many fast updating methods and parallel algorithms cannot be used.

DQMC Parallelization

Algorithmic approaches

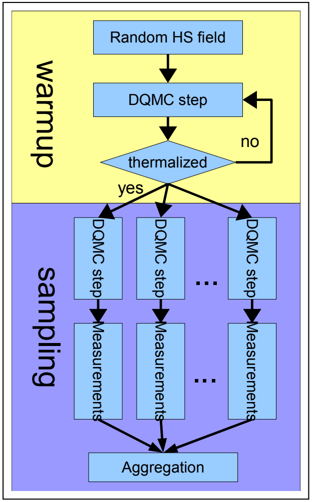
- Parallel Markov chain
- Rolling feeder algorithm
- Parallel matrix computations

System approaches

- Task decomposition
- Communication and computation overlapping
- Message compression
- Load balance

Parallel Markov Chain

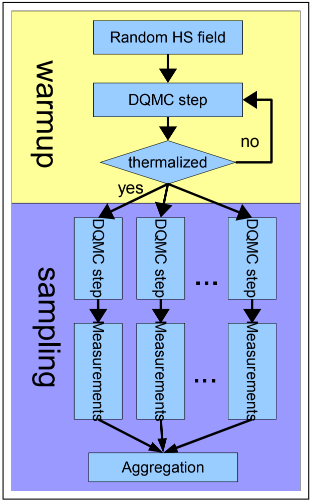
- The sampling stage can be parallelized embarrassingly.



Parallel Markov Chain

- The sampling stage can be parallelized embarrassingly.
- The speedup of parallelization is limited by the time of the warmup stage. (Amdahl's law)

$$\rho_{\text{speedup}} = \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}} + T_{\text{sampling}}/N_p} < \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}}}$$



Green's Function Calculation

Matrix G_k need be computed cyclically with B_{k-1} updated.

- $G_1 = (I + B_1 B_2 \cdots B_{L-1} B_L)^{-1}$.

Green's Function Calculation

Matrix G_k need be computed cyclically with B_{k-1} updated.

- $G_1 = (I + B_1 B_2 \cdots B_{L-1} B_L)^{-1}$.
- $G_2 = (I + B_2 B_3 \cdots B_L B_1)^{-1}$.

Green's Function Calculation

Matrix G_k need be computed cyclically with B_{k-1} updated.

- $G_1 = (I + B_1 B_2 \cdots B_{L-1} B_L)^{-1}$.
- $G_2 = (I + B_2 B_3 \cdots B_L B_1)^{-1}$.
- $G_3 = (I + B_3 B_4 \cdots B_1 B_2)^{-1}$.

Green's Function Calculation

Matrix G_k need be computed cyclically with B_{k-1} updated.

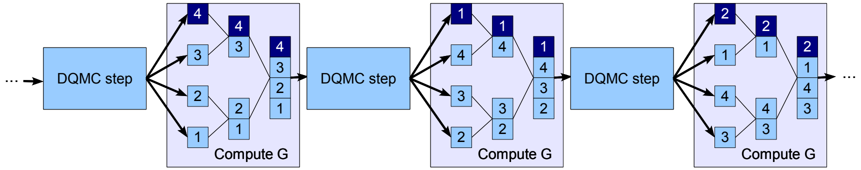
- $G_1 = (I + B_1 B_2 \cdots B_{L-1} B_L)^{-1}$.
- $G_2 = (I + B_2 B_3 \cdots B_L B_1)^{-1}$.
- $G_3 = (I + B_3 B_4 \cdots B_1 B_2)^{-1}$.
- $\cdots$

Green's Function Calculation

Matrix G_k need be computed cyclically with B_{k-1} updated.

- $G_1 = (I + B_1 B_2 \cdots B_{L-1} B_L)^{-1}$.
- $G_2 = (I + B_2 B_3 \cdots B_L B_1)^{-1}$.
- $G_3 = (I + B_3 B_4 \cdots B_1 B_2)^{-1}$.
- $\cdots$

Parallel reduction (takes $O(N^3 \log L)$ time.)

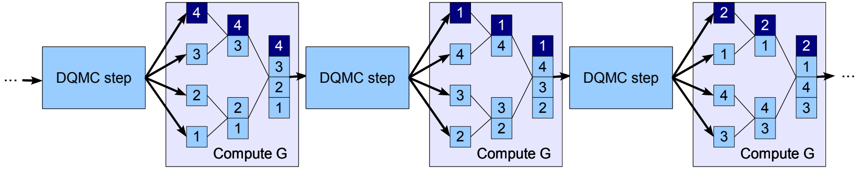


Green's Function Calculation

Matrix G_k need be computed cyclically with B_{k-1} updated.

- $G_1 = (I + B_1 B_2 \cdots B_{L-1} B_L)^{-1}$.
- $G_2 = (I + B_2 B_3 \cdots B_L B_1)^{-1}$.
- $G_3 = (I + B_3 B_4 \cdots B_1 B_2)^{-1}$.
- $\cdots$

Parallel reduction (takes $O(N^3 \log L)$ time.)



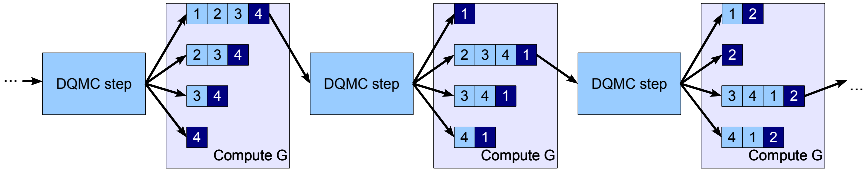
Numerically unstable!

Rolling Feeder Algorithm

The matrix product can be stably computed sequentially.

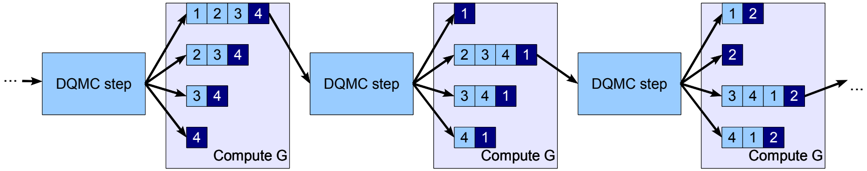
Rolling Feeder Algorithm

The matrix product can be stably computed sequentially.



Rolling Feeder Algorithm

The matrix product can be stably computed sequentially.



Tasks to get one G_k	Sequential	Parallel reduction	Rolling feeder
1. Matrix multiplication	L	$\log L$	1
2. Stabilization step	$O(L)$	$O(\log L)$	1
3. Inverting $(I + B_1 \dots B_L)$	1	1	1
4. Data transmission	N^2	$O(LN^2)$	N^2

Comparisons on resources and stability

Processor	$O(1)$	$O(L)$	$O(L)$
Numerically stable	Y	N	Y

Parallel Matrix Computations

Two matrix computation kernels are parallelized.

Parallel Matrix Computations

Two matrix computation kernels are parallelized.

1. The unequal time Green's function is computed by blocks in parallel

$$G^{\tau}_{k,\ell} = \begin{cases} (I + B_k \cdots B_1 B_L \cdots B_{k+1})^{-1} B_k \cdots B_{\ell+1} & k > \ell \\ (I + B_k \cdots B_1 B_L \cdots B_{k+1})^{-1} & k = \ell \\ -(I + B_k \cdots B_{k+1})^{-1} B_k \cdots B_1 B_L \cdots B_{\ell+1} & k < \ell \end{cases}$$

Parallel Matrix Computations

Two matrix computation kernels are parallelized.

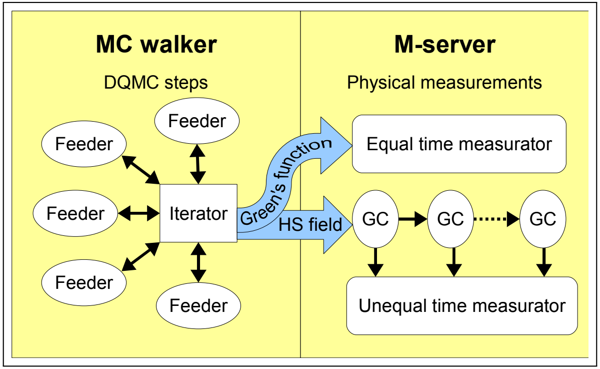
1. The unequal time Green's function is computed by blocks in parallel

$$G^{\tau}_{k,\ell} = \begin{cases} (I + B_k \cdots B_1 B_L \cdots B_{k+1})^{-1} B_k \cdots B_{\ell+1} & k > \ell \\ (I + B_k \cdots B_1 B_L \cdots B_{k+1})^{-1} & k = \ell \\ -(I + B_k \cdots B_{k+1})^{-1} B_k \cdots B_1 B_L \cdots B_{\ell+1} & k < \ell \end{cases}$$

2. The matrix-matrix multiplication of G_k and each block matrix of G^{τ} is speeded up using multicore.
   - The matrix size of G_k, 100-1000, is too small such that the matrix computation cannot be benefited by using MPI-style parallelization.

System Design

- The system contains several “simulators” for parallel Markov chain.
- Each simulator consists of a “walker” and a “M-server”.

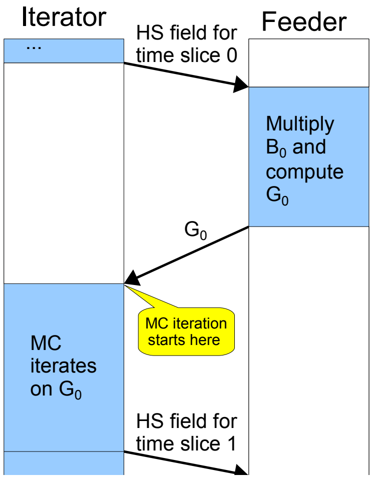


Implementation Techniques

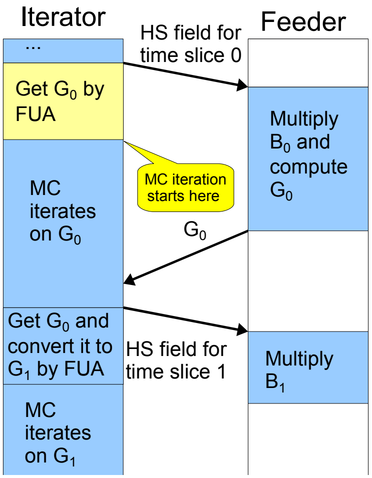
System is implemented for hybrid systems (cluster+multicore)

Task	MPI	OpenMP	Comm/comp overlapping	Message compression	Load balance
Parallel Markov chain	✓				
Rolling feeder algorithm	✓	✓	✓	✓	✓
Unequal time Green's fn	✓	✓		✓	✓
Physical measurement	✓			✓	✓

Communication/Computation Overlapping


Iterator
Feeder
...
HS field for time slice 0
Multiply B_0 and compute G_0
G_0
MC iteration starts here
MC iterates on G_0
HS field for time slice 1

Without overlapping


Iterator
Feeder
...
HS field for time slice 0
Get G_0 by FUA
Multiply B_0 and compute G_0
MC iteration starts here
MC iterates on G_0
G_0
Get G_0 and convert it to G_1 by FUA
HS field for time slice 1
Multiply B_1
MC iterates on G_1

Using fast update algorithm (FUA) to reduce waiting time

Load Balance

- Iterators are fully occupied $\rightarrow$ the bottleneck of speedup.

Load Balance

- Iterators are fully occupied $\rightarrow$ the bottleneck of speedup.
- Processor utilization can be enhanced by merging tasks.
  - For example, when computing unequal time Green's function, each processor can take care of more than one block submatrix.

Load Balance

- Iterators are fully occupied $\rightarrow$ the bottleneck of speedup.
- Processor utilization can be enhanced by merging tasks.
    - For example, when computing unequal time Green's function, each processor can take care of more than one block submatrix.
- The load balance problem: how many block submatrices should one processor compute?

Load Balance

- Iterators are fully occupied $\rightarrow$ the bottleneck of speedup.
- Processor utilization can be enhanced by merging tasks.
  - For example, when computing unequal time Green's function, each processor can take care of more than one block submatrix.
- The load balance problem: how many block submatrices should one processor compute?
- Using the queueing theory (Little's law) to estimate.

$$n_C = \max_{P \leq 1} \left\lfloor \frac{P}{\lambda T} \right\rfloor \leq \left\lfloor \frac{1}{\lambda T} \right\rfloor .$$

  - λ: arrival rate; T: processing time; P: processor utilization.

System and Benchmark

System

- Run on the IBM Blue Gene/P
- Each compute node is equipped with 850MHz PowerPC 450 quad-core processor and 2GB memory.
- IBM XL compilers with IBM BLAS and LAPACK libraries.

System and Benchmark

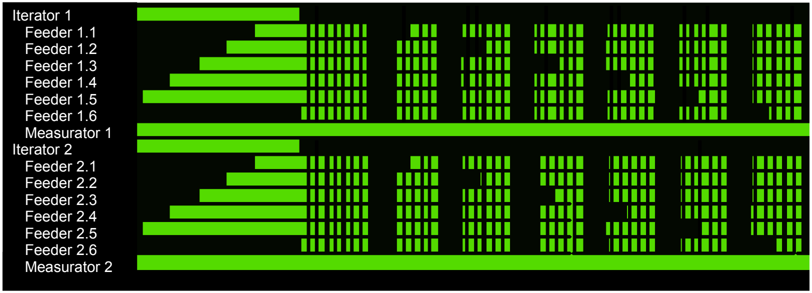
System

- Run on the IBM Blue Gene/P
- Each compute node is equipped with 850MHz PowerPC 450 quad-core processor and 2GB memory.
- IBM XL compilers with IBM BLAS and LAPACK libraries.

Benchmark

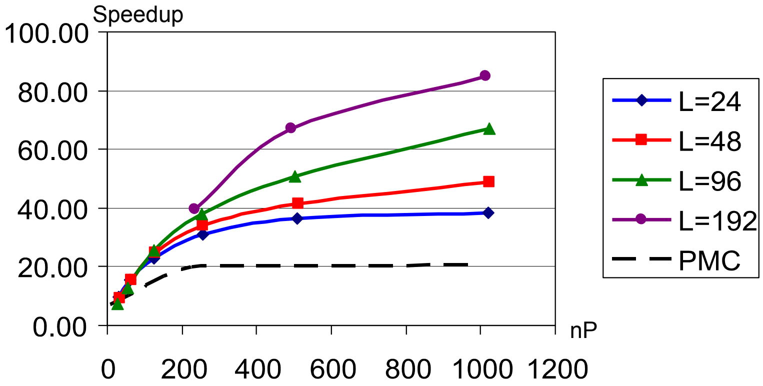
- DQMC simulation on a two-dimensional periodic lattice.
- The lattice size is $N = 16 \times 16 = 256$.
- The ratio of DQMC steps for the warmup stage and the sampling stages is $1 : 20$.

Communication Pattern

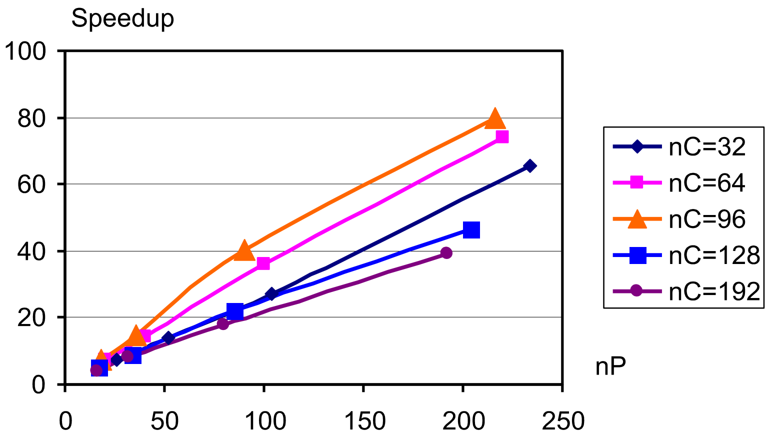


- Green bands show the waiting time of MPI_RECV.
- Iterators are fully occupied after started.

Speedup for Different *L*



Effect of Load Balance ($L = 96$)



- nC: number of block submatrices computed per processor.

Summary

- DQMC simulation for strongly correlated materials is a computationally intensive task, which is eager for parallelization.

Summary

- DQMC simulation for strongly correlated materials is a computationally intensive task, which is eager for parallelization.
- We targeted the hybrid massive parallel systems, and explored the parallelism of DQMC simulations on different levels of granularity.

Summary

- DQMC simulation for strongly correlated materials is a computationally intensive task, which is eager for parallelization.
- We targeted the hybrid massive parallel systems, and explored the parallelism of DQMC simulations on different levels of granularity.
- Our implementation shows over 80x speedup on thousand processors, which is much better than embarrassing parallelization (speedup < 21).

Future Works

- More fine-grain parallel matrix computation kernels (pivoted QR, QR, matrix inversion) to fully utilize the computational power of multicores.

Future Works

- More fine-grain parallel matrix computation kernels (pivoted QR, QR, matrix inversion) to fully utilize the computational power of multicores.
- Better system design to enhance the processor utilization.

Future Works

- More fine-grain parallel matrix computation kernels (pivoted QR, QR, matrix inversion) to fully utilize the computational power of multicores.
- Better system design to enhance the processor utilization.
- Different physics models and methods.

Future Works

- More fine-grain parallel matrix computation kernels (pivoted QR, QR, matrix inversion) to fully utilize the computational power of multicores.
- Better system design to enhance the processor utilization.
- Different physics models and methods.
- Code is still in the experimental stage. Further development is required for practical use.