

# Parallelizing the Hamiltonian Computation in DQMC Simulations: Checkerboard Method for Sparse Matrix Exponentials on Multicore and GPU

Che-Rung Lee

National Tsing Hua University

joint work with Zhi-Hung Chen and Quey-Liang Kao

Second International Workshop on  
Accelerators and Hybrid Exascale Systems (AsHES)  
May 25th, 2012

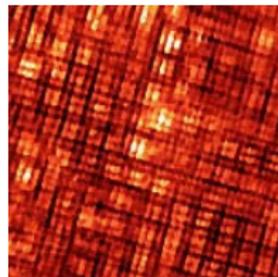


# Outline

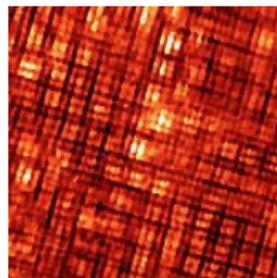
- 1 Determinant quantum Monte Carlo simulations
- 2 Matrix multiplication of sparse matrix exponentials
- 3 Parallel block checkerboard methods on multicore and GPU
- 4 Experiments and results
- 5 Concluding remarks



To study the properties of solid-state materials:  
magnetism, metal-insulator transition, high  
temperature superconductivity, ...



To study the properties of solid-state materials:  
magnetism, metal-insulator transition, high  
temperature superconductivity, ...

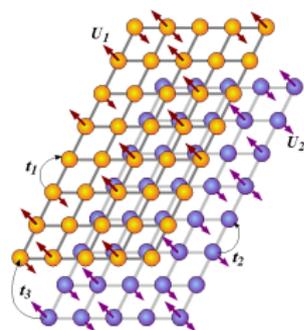


The Hubbard model:

- Energy operator  $H$  is associated with a lattice of particles.
- Boltzmann weight is expressed as a path integral

$$e^{-\beta H} \approx e^{-\tau H(h_1)} e^{-\tau H(h_2)} \dots e^{-\tau H(h_L)}.$$

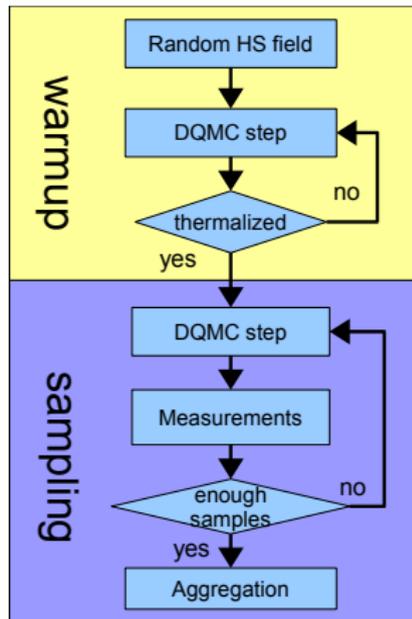
- $\beta = 1/T$  is the “imaginary time”.
- $\tau = \beta/L$  is the discretized time step.
- $\{h_i\}$  is the “Hubbard-Stratonovich field”.



# Determinant Quantum Monte Carlo (DQMC) Simulations

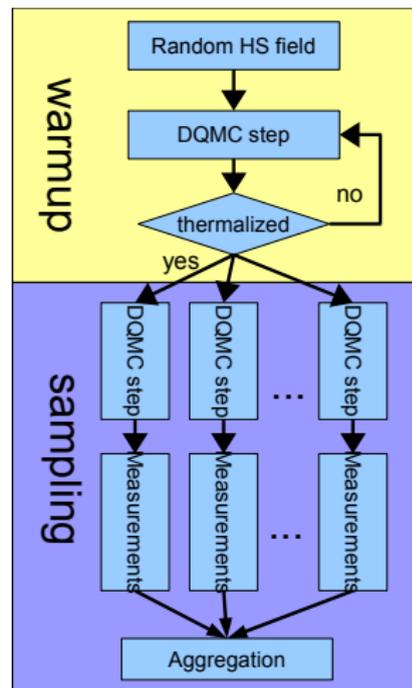
## DQMC algorithm

- 1 Given a random  $h = (h_{\ell,i}) = (\pm 1)$ .
- 2 Until there are enough measurements  
For  $\ell = 1, \dots, L$  and  $i = 1, \dots, N$ 
  - 1 Propose a new HS config  $h'$ .
  - 2 Compute the ratio  $\gamma$  of the determinants of new/old configs.
  - 3 Generate a random number  $\rho \in [0, 1]$ .
  - 4 If  $\gamma > \rho$ , accept  $h = h'$ .
  - 5 If the system is thermalized, sample the interested physical measurements.
- 3 Aggregate the sampled measurements.



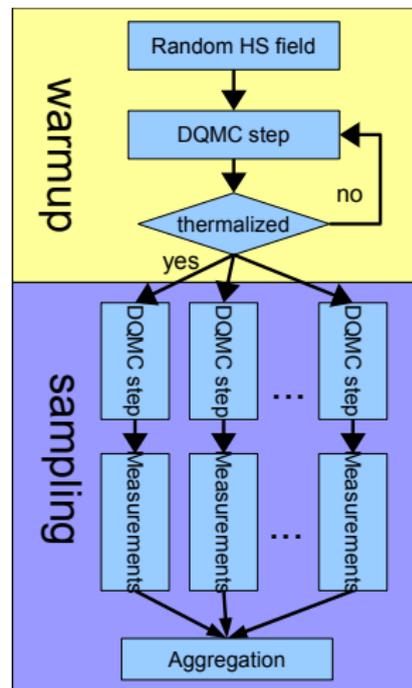
# DQMC Parallelization

- Parallel Monte Carlo method can speedup DQMC simulations by parallelizing the sampling stage.



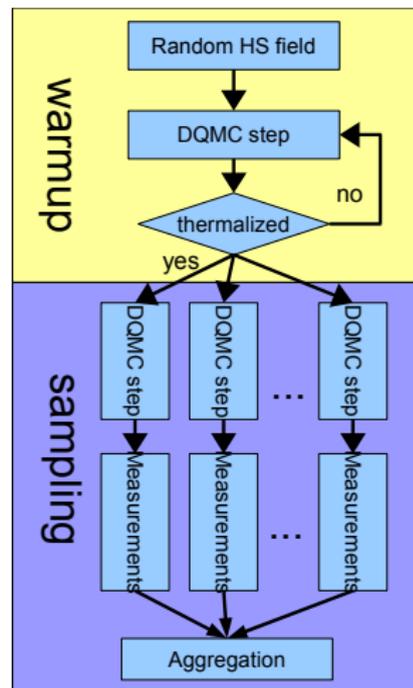
# DQMC Parallelization

- Parallel Monte Carlo method can speedup DQMC simulations by parallelizing the sampling stage.
- Coarse-grained parallelization. (Communication only happens before sampling and in aggregation.)



# DQMC Parallelization

- Parallel Monte Carlo method can speedup DQMC simulations by parallelizing the sampling stage.
- Coarse-grained parallelization. (Communication only happens before sampling and in aggregation.)
- Strong scalability if the number of desired samplings is much larger than the number of processors.



# Computational Challenges

- By Amdahl's law, the speedup of parallel Monte Carlo method is limited by the warmup stage (non-parallelizable).

$$\begin{aligned}\text{Speedup} &= \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}} + T_{\text{sampling}}/p} \\ &\rightarrow \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}}}\end{aligned}$$



# Computational Challenges

- By Amdahl's law, the speedup of parallel Monte Carlo method is limited by the warmup stage (non-parallelizable).

$$\begin{aligned}\text{Speedup} &= \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}} + T_{\text{sampling}}/p} \\ &\rightarrow \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}}}\end{aligned}$$

- Parallel Monte Carlo method does not scale with problem size, i.e. number of particles and discretized time length.



# Computational Challenges

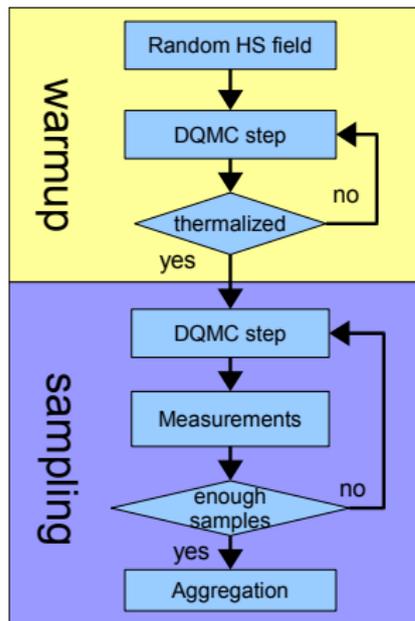
- By Amdahl's law, the speedup of parallel Monte Carlo method is limited by the warmup stage (non-parallelizable).

$$\begin{aligned}\text{Speedup} &= \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}} + T_{\text{sampling}}/p} \\ &\rightarrow \frac{T_{\text{warmup}} + T_{\text{sampling}}}{T_{\text{warmup}}}\end{aligned}$$

- Parallel Monte Carlo method does not scale with problem size, i.e. number of particles and discretized time length.
- Coarse grained parallelization does not fit well on multicore and GPU.
  - The computation of each DQMC step is complicated.
  - Slower execution because of resource contention.
  - Memory per core is reduced with the number of cores.



# Inside Each DQMC Step

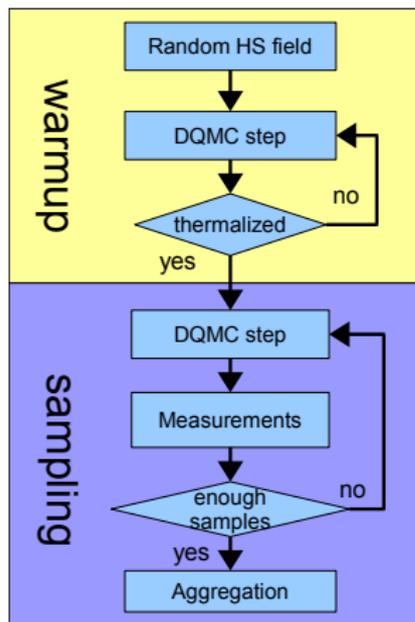


## A DQMC step

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



# Inside Each DQMC Step



## A DQMC step

- 1 Propose a local change:  $h \rightarrow h'$ .
- 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 Kernel: Green's function calculation

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

for computation of  $\det(e^{-\beta H(h')})$  and physical measurements.



# Green's Function Calculation

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

- $N$ : the number of particles;  $L$ : the number of time slices.



# Green's Function Calculation

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

- $N$ : the number of particles;  $L$ : the number of time slices.
- Time complexity of computing  $G$  is  $O(N^3 L)$ .



# Green's Function Calculation

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

- $N$ : the number of particles;  $L$ : the number of time slices.
- Time complexity of computing  $G$  is  $O(N^3 L)$ .
- For  $10^3$  warmup steps and  $10^4$  sampling steps, it takes 15 hours.
  - For large simulations,  $N = O(10^4)$ ,  $L = O(10^2)$ , the projected execution time could take several days to months.



# Green's Function Calculation

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

- $N$ : the number of particles;  $L$ : the number of time slices.
- Time complexity of computing  $G$  is  $O(N^3 L)$ .
- For  $10^3$  warmup steps and  $10^4$  sampling steps, it takes 15 hours.
  - For large simulations,  $N = O(10^4)$ ,  $L = O(10^2)$ , the projected execution time could take several days to months.
- Profile of a DQMC simulation ( $N = 256$ ,  $L = 96$ )

Matrix kernel	Execution time
Matrix-matrix multiplication	72.39%
Pivoted QR decomposition	17.83%
Matrix inversion	3.02%
Others	6.76%



# Green's Function Calculation

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

- $N$ : the number of particles;  $L$ : the number of time slices.
- Time complexity of computing  $G$  is  $O(N^3 L)$ .
- For  $10^3$  warmup steps and  $10^4$  sampling steps, it takes 15 hours.
  - For large simulations,  $N = O(10^4)$ ,  $L = O(10^2)$ , the projected execution time could take several days to months.
- Profile of a DQMC simulation ( $N = 256$ ,  $L = 96$ )

Matrix kernel	Execution time
Matrix-matrix multiplication	72.39%
Pivoted QR decomposition	17.83%
Matrix inversion	3.02%
Others	6.76%



# Outline

- 1 Determinant quantum Monte Carlo simulations
- 2 Matrix multiplication of sparse matrix exponentials**
- 3 Parallel block checkerboard methods on multicore and GPU
- 4 Experiments and results
- 5 Concluding remarks



# Matrix-matrix Multiplication

- Some tuned result on multicore and on GPU (Fermi)
  - DGEMM on Intel Core i7-920 4 core with MKL is about 40 Gflop/s. (my laptop.)
  - SGEMM can reach 662 Gflop/s on Fermi. [Jakub Kurzak LAWN 245, 2010]
  - DGEMM (362Gflop/s on Fermi) [Guangming Tan et. al. SC11]



# Matrix-matrix Multiplication

- Some tuned result on multicore and on GPU (Fermi)
  - DGEMM on Intel Core i7-920 4 core with MKL is about 40 Gflop/s. (my laptop.)
  - SGEMM can reach 662 Gflop/s on Fermi. [Jakub Kurzak LAWN 245, 2010]
  - DGEMM (362Gflop/s on Fermi) [Guangming Tan et. al. SC11]
- It is great, but the running time grows cubically with problem size  $N$ .



# Matrix-matrix Multiplication

- Some tuned result on multicore and on GPU (Fermi)
  - DGEMM on Intel Core i7-920 4 core with MKL is about 40 Gflop/s. (my laptop.)
  - SGEMM can reach 662 Gflop/s on Fermi. [Jakub Kurzak LAWN 245, 2010]
  - DGEMM (362Gflop/s on Fermi) [Guangming Tan et. al. SC11]
- It is great, but the running time grows cubically with problem size  $N$ .
- Sparse-dense matrix multiplication takes only  $O(N^2)$  time.



# Matrix-matrix Multiplication

- Some tuned result on multicore and on GPU (Fermi)
  - DGEMM on Intel Core i7-920 4 core with MKL is about 40 Gflop/s. (my laptop.)
  - SGEMM can reach 662 Gflop/s on Fermi. [Jakub Kurzak LAWN 245, 2010]
  - DGEMM (362Gflop/s on Fermi) [Guangming Tan et. al. SC11]
- It is great, but the running time grows cubically with problem size  $N$ .
- Sparse-dense matrix multiplication takes only  $O(N^2)$  time.
- In the Green's function calculation,  $G = (I + B_L B_{L-1} \cdots B_2 B_1)^{-1}$ , each  $B_i = e^A$  is a matrix exponential.

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{A^k}{k!}.$$



# Matrix-matrix Multiplication

- Some tuned result on multicore and on GPU (Fermi)
  - DGEMM on Intel Core i7-920 4 core with MKL is about 40 Gflop/s. (my laptop.)
  - SGEMM can reach 662 Gflop/s on Fermi. [Jakub Kurzak LAWN 245, 2010]
  - DGEMM (362Gflop/s on Fermi) [Guangming Tan et. al. SC11]
- It is great, but the running time grows cubically with problem size  $N$ .
- Sparse-dense matrix multiplication takes only  $O(N^2)$  time.
- In the Green's function calculation,  $G = (I + B_L B_{L-1} \cdots B_2 B_1)^{-1}$ , each  $B_i = e^A$  is a matrix exponential.

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{A^k}{k!}.$$

- Matrix  $A$  is highly sparse, but  $e^A$  is dense.



# Checkerboard Method

Checkerboard method can approximate  $e^A$  by  $e^A \approx e^{A_1} e^{A_2} \dots e^{A_k}$ , in which each  $e^{A_j}$  is sparse.



# Checkerboard Method

Checkerboard method can approximate  $e^A$  by  $e^A \approx e^{A_1} e^{A_2} \dots e^{A_k}$ , in which each  $e^{A_j}$  is sparse.

## Theorem

If  $A_i$  is *strictly sparse* with zero diagonal,  $e^{A_i}$  has the same sparse pattern as  $A_i$  with diagonal fill in.



# Checkerboard Method

Checkerboard method can approximate  $e^A$  by  $e^A \approx e^{A_1} e^{A_2} \dots e^{A_k}$ , in which each  $e^{A_j}$  is sparse.

## Theorem

If  $A_i$  is *strictly sparse* with zero diagonal,  $e^{A_i}$  has the same sparse pattern as  $A_i$  with diagonal fill in.

## Definition (strictly sparse)

A matrix is strictly sparse if it contains at most one nonzero per row and per column.



# Checkerboard Method

Checkerboard method can approximate  $e^A$  by  $e^A \approx e^{A_1} e^{A_2} \dots e^{A_k}$ , in which each  $e^{A_j}$  is sparse.

## Theorem

If  $A_i$  is *strictly sparse* with zero diagonal,  $e^{A_i}$  has the same sparse pattern as  $A_i$  with diagonal fill in.

## Definition (strictly sparse)

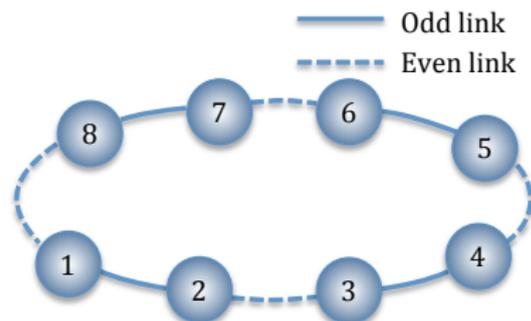
A matrix is strictly sparse if it contains at most one nonzero per row and per column.

## Checkerboard method for computing $e^A$

- 1 Split  $A = A_1 + A_2 + \dots + A_k$  such that each  $A_i$  is strictly sparse.
- 2 Exponentiate each  $A_i$ .
- 3 Return  $e^{A_1} e^{A_2} \dots e^{A_k}$  as an approximation to  $e^A$ .

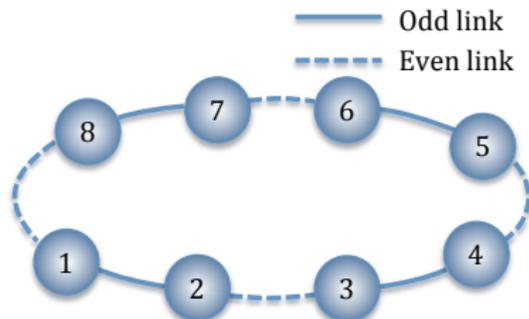
# Example: One Dimensional Ring

$$A = \begin{bmatrix} 0 & h & & h \\ h & 0 & & \\ & & \ddots & \ddots \\ & & & 0 & h \\ h & & h & & 0 \end{bmatrix}$$



# Example: One Dimensional Ring

$$A = \begin{bmatrix} 0 & h & & h \\ h & 0 & & \\ & & \ddots & \ddots \\ & & & 0 & h \\ h & & & h & 0 \end{bmatrix}$$



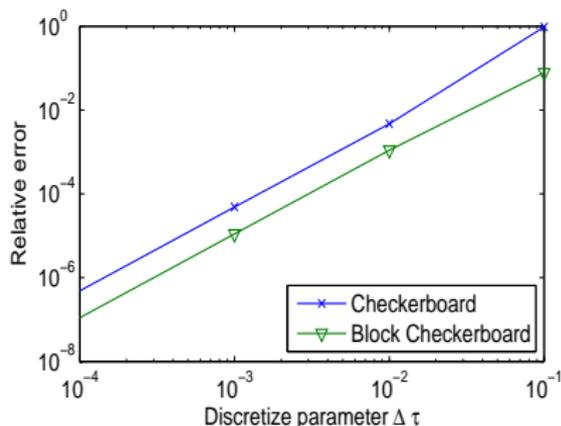
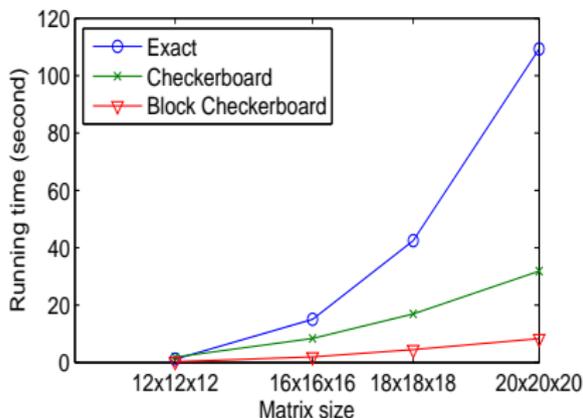
$$e^A \approx \begin{bmatrix} D & & \\ & \ddots & \\ & & D \end{bmatrix} \begin{bmatrix} \cosh(h) & & \sinh(h) \\ & D & \\ \sinh(h) & & \cosh(h) \end{bmatrix}$$

where  $D = \begin{bmatrix} \cosh(h) & \sinh(h) \\ \sinh(h) & \cosh(h) \end{bmatrix}$ .



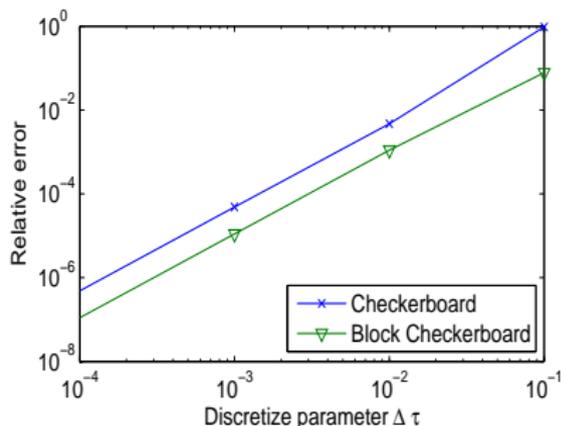
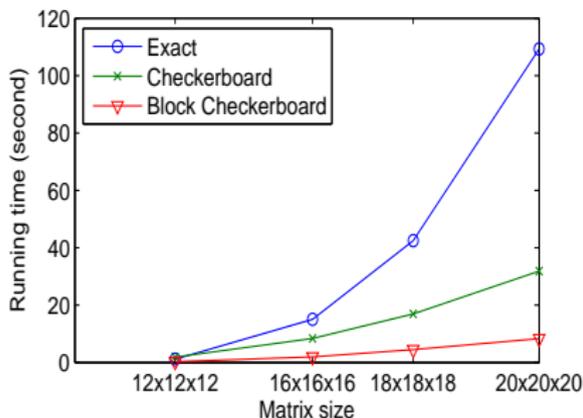
# Block Checkerboard Method

- Exploring block structure of sparse matrices can obtain better performance and accuracy.



# Block Checkerboard Method

- Exploring block structure of sparse matrices can obtain better performance and accuracy.

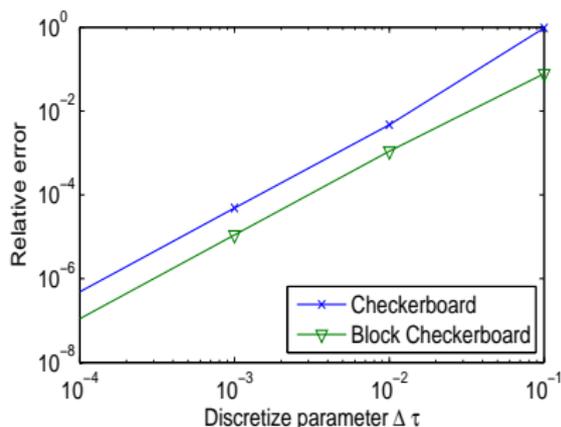
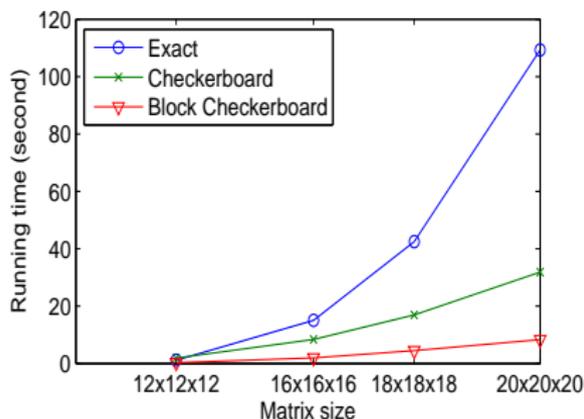


- The algorithm is similar, but the basic element is a block submatrix.



# Block Checkerboard Method

- Exploring block structure of sparse matrices can obtain better performance and accuracy.



- The algorithm is similar, but the basic element is a block submatrix.
- We will focus on the parallelization of block checkerboard method.



# Outline

- 1 Determinant quantum Monte Carlo simulations
- 2 Matrix multiplication of sparse matrix exponentials
- 3 Parallel block checkerboard methods on multicore and GPU**
- 4 Experiments and results
- 5 Concluding remarks



# Computational Difficulties

- Combination of sparse matrix and dense matrix computation.  
(generalized SPMV)



- Combination of sparse matrix and dense matrix computation.  
(generalized SPMV)



Dense matrix-matrix  
multiplication  
Computational bound

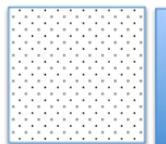


# Computational Difficulties

- Combination of sparse matrix and dense matrix computation.  
(generalized SPMV)



Dense matrix-matrix  
multiplication  
Computational bound



Sparse matrix-vector  
multiplication  
Memory bound

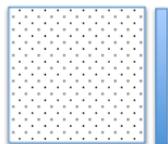


# Computational Difficulties

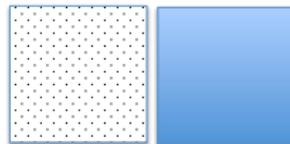
- Combination of sparse matrix and dense matrix computation.  
(generalized SPMV)



Dense matrix-matrix  
multiplication  
Computational bound



Sparse matrix-vector  
multiplication  
Memory bound



Sparse matrix-matrix  
multiplication  
?

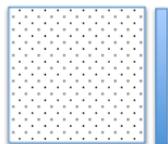


# Computational Difficulties

- Combination of sparse matrix and dense matrix computation.  
(generalized SPMV)



Dense matrix-matrix  
multiplication  
Computational bound



Sparse matrix-vector  
multiplication  
Memory bound



Sparse matrix-matrix  
multiplication  
?

- Multiplication of a sequence of sparse matrices of different characteristics.

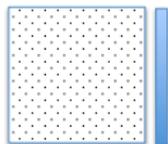


# Computational Difficulties

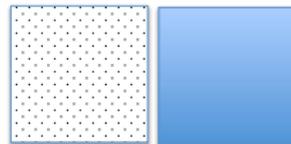
- Combination of sparse matrix and dense matrix computation.  
(generalized SPMV)



Dense matrix-matrix  
multiplication  
Computational bound



Sparse matrix-vector  
multiplication  
Memory bound



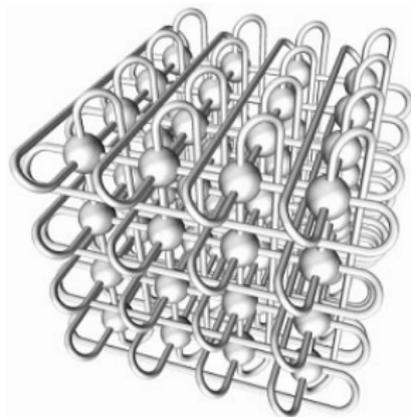
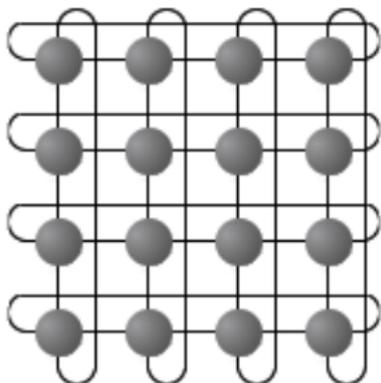
Sparse matrix-matrix  
multiplication  
?

- Multiplication of a sequence of sparse matrices of different characteristics.
- Matrix size is not large enough to reach good performance.



# 2D and 3D Torus Lattices

- The kinetic matrices in our study are 2D and 3D torus lattices.



Images are from <http://www.trampelwurm.ch/schmidt/wilhelmtux/swissremix/html/Linuxfibel/netstruct.htm> and <http://www.fujitsu.com/global/news/pr/archives/month/2009/20090717-01.html>

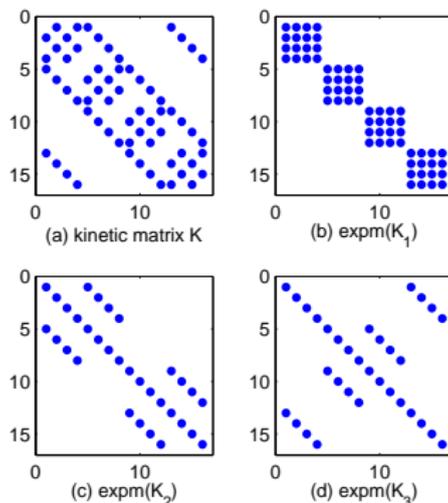


# 2D Lattice and Kinetic Matrix

- The kinetic matrix of 2D lattices can be split into 3 block strictly sparse matrices,  $K_1$ ,  $K_2$ , and  $K_3$ .
- Checkerboard method approximates the matrix exponential by

$$e^{K_3/2} e^{K_2/2} e^{K_1} e^{K_2/2} e^{K_3/2}.$$

- The figure shows the sparse pattern of the matrix of a  $4 \times 4$  2D lattice is shown, and the matrix exponentials,  $e^{K_1}$ ,  $e^{K_2}$ , and  $e^{K_3}$ .



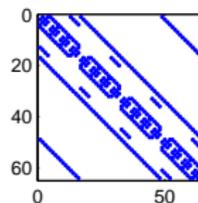
# 3D Lattice and Kinetic Matrix

- The kinetic matrix of 3D lattices can be split into 5 block strictly sparse matrices,  $K_1, K_2, K_3, K_4$ , and  $K_5$ .

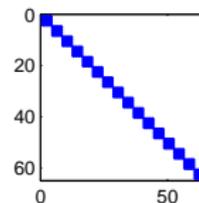
- Checkerboard method approximates the matrix exponential by

$$e^{\frac{K_5}{2}} e^{\frac{K_4}{2}} e^{\frac{K_3}{2}} e^{\frac{K_2}{2}} e^{K_1} e^{\frac{K_5}{2}} e^{\frac{K_3}{2}} e^{\frac{K_4}{2}} e^{\frac{K_5}{2}}.$$

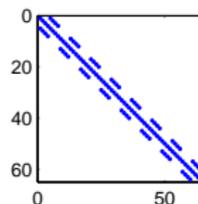
- The figure shows the sparse pattern of the matrix of a  $4 \times 4$  2D lattice is shown, and the matrix exponentials,  $e^{K_1}, e^{K_2}, e^{K_3}, e^{K_4}$ , and  $e^{K_5}$ .



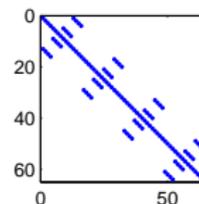
(a) kinetic matrix K



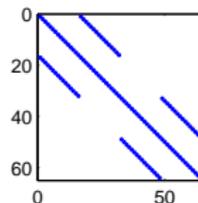
(b)  $\text{expm}(K_1)$



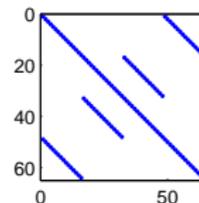
(c)  $\text{expm}(K_2)$



(d)  $\text{expm}(K_3)$



(e)  $\text{expm}(K_4)$



(f)  $\text{expm}(K_5)$



# $e^{K_1}$ : Block Diagonal Matrix

In 2D and 3D problems,  $e^{K_1} = \begin{pmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_k \end{pmatrix}$ , block diagonal.



# $e^{K_1}$ : Block Diagonal Matrix

In 2D and 3D problems,  $e^{K_1} = \begin{pmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_k \end{pmatrix}$ , block diagonal.

For  $B = \begin{pmatrix} B_{11} & B_{12} & \dots & B_{1k} \\ B_{21} & B_{22} & \dots & B_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ B_{k1} & B_{k2} & \dots & B_{kk} \end{pmatrix}$ , the product of  $e^{K_1} B$  is

$$e^{K_1} B = \begin{pmatrix} A_1 B_{11} & A_1 B_{12} & \dots & A_1 B_{1k} \\ A_2 B_{21} & A_2 B_{22} & \dots & A_2 B_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ A_k B_{k1} & A_k B_{k2} & \dots & A_k B_{kk} \end{pmatrix}.$$








# Parallelize Algorithms for $e^{K_1 B}$

Different algorithms need be designed from different types of matrices.



# Parallelize Algorithms for $e^{K_1} B$

Different algorithms need be designed from different types of matrices.  
Block based algorithm for  $e^{K_1} B$ .

## Parallelization of $e^{K_1} B$

For each  $B_{ij}$  do

Parallel compute  $C_{i,j} = A_i B_{ij}$ .

End for each



# Parallelize Algorithms for $e^{K_1} B$

Different algorithms need be designed from different types of matrices.  
Block based algorithm for  $e^{K_1} B$ .

## Parallelization of $e^{K_1} B$

```
For each  $B_{ij}$  do
    Parallel compute  $C_{i,j} = A_i B_{ij}$ .
End for each
```

For cache effect, finer grained parallelization is needed.

## Finer-grained parallelization of $e^{K_1} B$

```
For each  $B_{ij}$  do
    For each sub-block of  $C_{ij}$ :  $C_{ij}(k, \ell)$  do
        Parallel compute  $C_{i,j}(k, \ell)$ .
    End for each
End for each
```

# Parallelize Algorithms for $e^{K_i}B, i \neq 1$

Column based parallelization

For better performance, aggregate the computation of  $e^{K_i}B$ .

## Parallelization of $e^{K_i}B, i \neq 1$

For each column  $b_i$  in  $B$  do

Parallel compute  $e^{K_2/2}e^{K_3/2}b_i$  or  $e^{K_2/2} \dots e^{K_5/2}b_i$ .

End for each



# Parallelize Algorithms for $e^{K_i}B, i \neq 1$

Column based parallelization

For better performance, aggregate the computation of  $e^{K_i}B$ .

## Parallelization of $e^{K_i}B, i \neq 1$

For each column  $b_i$  in  $B$  do

Parallel compute  $e^{K_2/2}e^{K_3/2}b_i$  or  $e^{K_2/2} \dots e^{K_5/2}b_i$ .

End for each

Divide  $b_i$  to fit cache

## Parallelization of $e^{K_i}B, i \neq 1$

For each column  $b_i$  in  $B$  do

For each segment of  $b_i(:)$  do

Parallel compute  $e^{K_2/2}e^{K_3/2}b_i(:)$  or  $e^{K_2/2} \dots e^{K_5/2}b_i(:)$ .

End for each

End for each

# Outline

- 1 Determinant quantum Monte Carlo simulations
- 2 Matrix multiplication of sparse matrix exponentials
- 3 Parallel block checkerboard methods on multicore and GPU
- 4 Experiments and results**
- 5 Concluding remarks



# Experimental Setting

## Experiment setting

- We experimented the parallel algorithms on multicore and GPU.
- Matrices are from 2D and 3D toruses.

## Multicore CPU

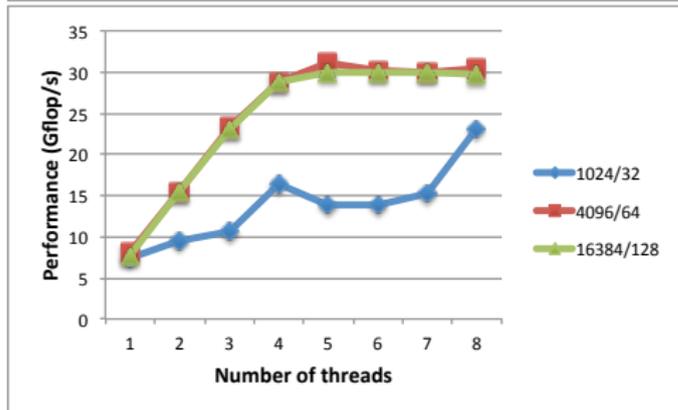
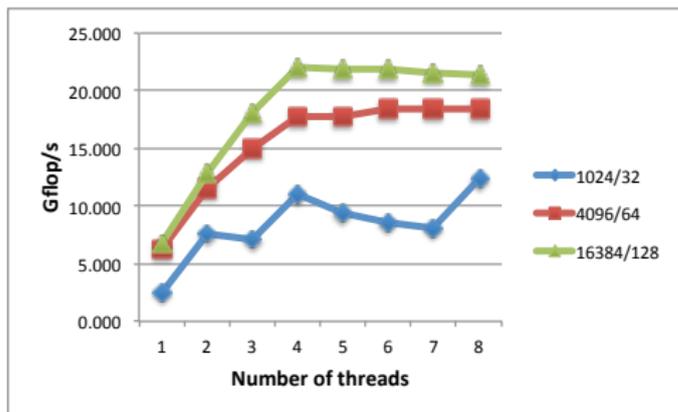
- Intel Core i7 950, 8G RAM, whose peak performance is 48.48Gflop/s.
- Using DGEMM in MKL 11 for block matrix multiplication and for comparison.

## GPU

- GeForce GTX 480, whose peak performance is 1344GFlop/s.
- Using CUDA 4 and DGEMM and SGEMM CUDA SDK for comparison.



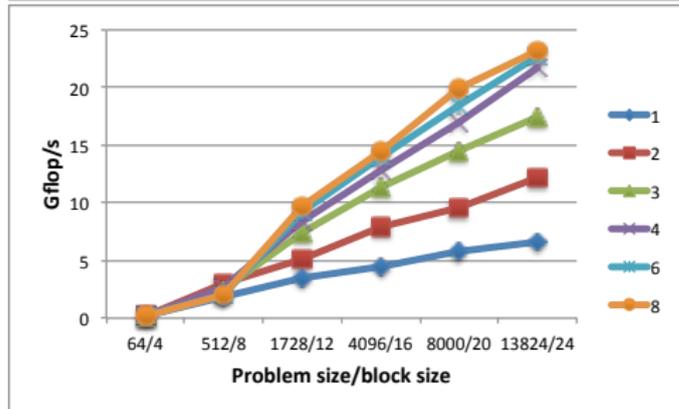
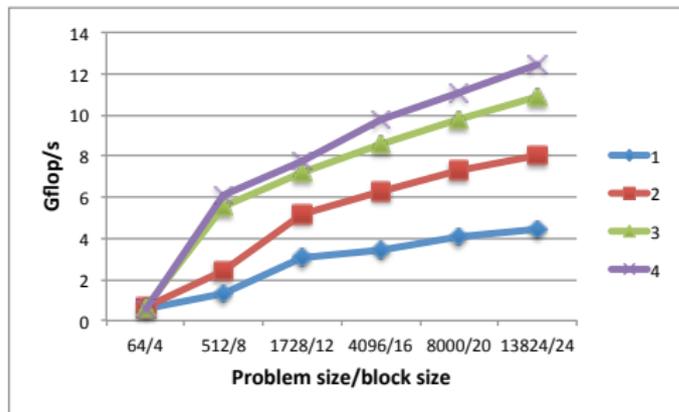
# Results of 2D Problems on Multicore



- Lattice size:  $32 \times 32$ ,  $64 \times 64$ , and  $128 \times 128$ .
- Larger problem has more stable performance result.
- The results of  $32 \times 32$  are less stable than those of other cases.
- Hyperthreading is almost no effect.



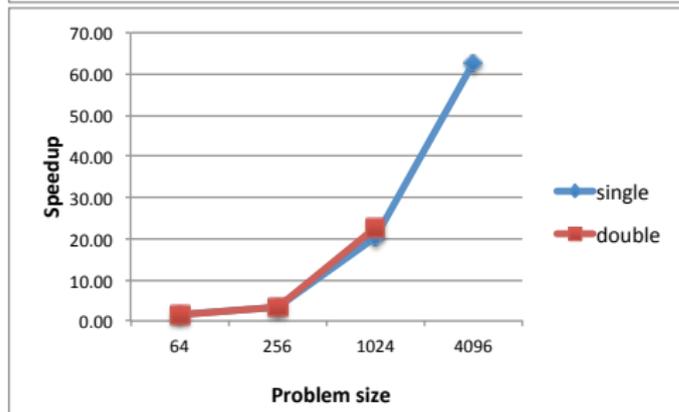
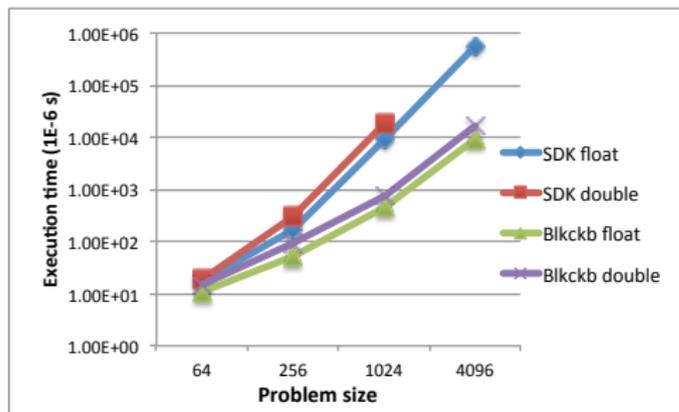
# Results of 3D Problems on Multicore



- Lattice size:  $4 \times 4 \times 4$ ,  $8 \times 8 \times 8$ ,  $12 \times 12 \times 12$ ,  $16 \times 16 \times 16$ ,  $20 \times 20 \times 20$ , and  $24 \times 24 \times 24$ .
- The speedup is about 3 using 4 cores.
- The Gflop/s of  $e^{K_1}$  is much better than that of the overall performance.
- Hyperthreading is working, but its effect is decreasing.



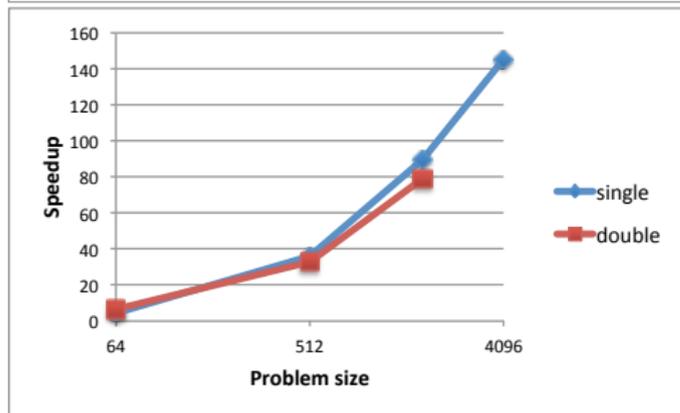
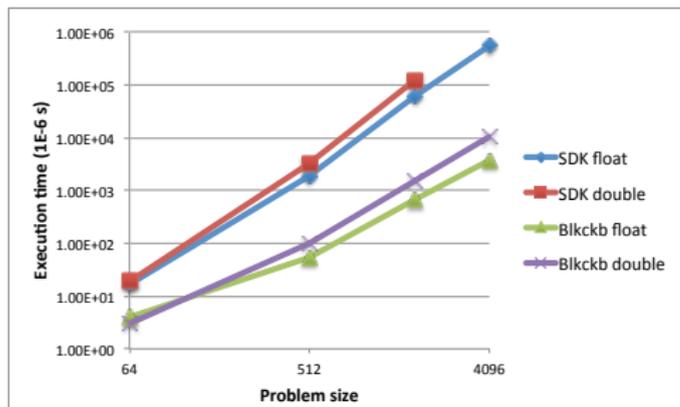
# Results of 2D Problems on GPU



- Lattice size:  $8 \times 8$ ,  $16 \times 16$ ,  $32 \times 32$ , and  $64 \times 64$ .
- The speedup is up to 67 for single precision.
- DGEMM of CUDA SDK cannot finish the  $64 \times 64$  case.



# Results of 3D Problems on GPU



- Lattice size:  $4 \times 4 \times 4$ ,  $8 \times 8 \times 8$ ,  $12 \times 12 \times 12$ , and  $16 \times 16 \times 16$ .
- The speedup is up to 147 for single precision.
- DGEMM of CUDA SDK cannot finish the  $16 \times 16 \times 16$  case.



# Outline

- 1 Determinant quantum Monte Carlo simulations
- 2 Matrix multiplication of sparse matrix exponentials
- 3 Parallel block checkerboard methods on multicore and GPU
- 4 Experiments and results
- 5 Concluding remarks



# Concluding Remarks

- Scientific applications request more and more computational power to enable larger and larger simulations. But to extract performance from modern HPC machines, which hybrid coarse-grained and fine-grained parallel architectures, application developers need to redesign the program.



# Concluding Remarks

- Scientific applications request more and more computational power to enable larger and larger simulations. But to extract performance from modern HPC machines, which hybrid coarse-grained and fine-grained parallel architectures, application developers need to redesign the program.
- For DQMC simulations, checkerboard method can make the algorithm more scalable with problem size. This paper presents the preliminary study of the parallelization of checkerboard method on multicore and GPU. Further performance tunings are required.



# Concluding Remarks

- Scientific applications request more and more computational power to enable larger and larger simulations. But to extract performance from modern HPC machines, which hybrid coarse-grained and fine-grained parallel architectures, application developers need to redesign the program.
- For DQMC simulations, checkerboard method can make the algorithm more scalable with problem size. This paper presents the preliminary study of the parallelization of checkerboard method on multicore and GPU. Further performance tunings are required.
- Parallelization of general lattice geometry of checkerboard method need be studied. The integration of the parallel checkerboard method to package needs be done.

