# Parallelization of DQMC Simulation for Strongly Correlated Electron Systems

Che-Rung Lee
Dept. of Computer Science
National Tsing-Hua University
Hsin-Chu, Taiwan
cherung@cs.nthu.edu.tw

I-Hsin Chung
IBM T.J. Watson Research Center
Yorktown Heights, NY USA
ihchung@us.ibm.com

Zhaojun Bai
Dept. of of Computer Science
University of California, Davis
Davis, CA USA
bai@cs.ucdavis.edu

***Abstract*—Determinant Quantum Monte Carlo (DQMC) simulation has been widely used to reveal macroscopic properties of strong correlated materials. However, parallelization of the DQMC simulation is extremely challenging duo to the serial nature of underlying Markov chain and numerical stability issues. We extend previous work with novelty by presenting a hybrid granularity parallelization (HGP) scheme that combines algorithmic and implementation techniques to speed up the DQMC simulation. From coarse-grained parallel Markov chain and task decompositions to fine-grained parallelization methods for matrix computations and Green's function calculations, the HGP scheme explores the parallelism on different levels and maps the underlying algorithms onto different computational components that are suitable for modern high performance heterogeneous computer systems. Practical techniques, such as communication and computation overlapping, message compression and load balancing are also considered in the proposed HGP scheme. We have implemented the DQMC simulation with the HGP scheme on an IBM Blue Gene/P system. The effectiveness of the new scheme is demonstrated through both theoretical analysis and performance results. Experiments have shown over a factor of 80 speedups on an IBM Blue Gene/P system with 1,014 computational processors.**

***Index Terms*—Hubbard model; Quantum Monte Carlo simulation; Heterogenenous system; Parallelization;**

## I. Introduction

Current study on complex materials has brought new impetus to develop high performance programs for condensed matter physics simulations. For instance, the study of strongly correlated materials in many technically important applications requires much larger simulation models to reveal long range correlations. Another example comes from the multi-scale physics simulation, where Hirsch-Fye QMC Quantum Monte Carlo (HF-QMC) algorithm is integrated into the simulation as a Dynamical Cluster Approximation (DCA) solver. Iterative executions of HF-QMC and DCA generate high demands for intensive computation. In [1], an implementation of combined HF-QMC and DCA algorithms, called DCA++, demonstrated high scalability of performance by reaching over 400 TFlop/s on a computer system with 49,288 cores. The DCA++ program explores three levels of parallelism. The first level distributes different disorder configurations to processors and aggregates the result at the end. The second level employs the technique of parallel Markov chain. Both the first and second levels use MPI for parallelization. The third level utilizes OpenMP method for matrix computations and physical measurement calculations.

Determinant QMC (DQMC) [2], [3], [4], [5] is similar to the Hirsch-Fye QMC (HF-QMC) in the DCA++ program. It is particularly suitable for simulating strongly correlated materials. In the DQMC simulation, the matrices are of the dimension equal to the number of spatial lattice sites $N$. They are dense and involve the product of $L$ sparse matrices, where $L$ is the number of time slices from the discretization of temporal domain. The computational complexity of the sequential code is at the order of $N^3L$. With the proper matrix representation, the DQMC simulation can be done on many hundreds of spatial sites. This ability makes the DQMC method particularly suitable for simulating strongly correlated materials, since long range correlations can only be revealed by large enough lattice sizes. QUantum Electron Simulation Toolbox (QUEST) is a Fortran 90/95 implementation of the DQMC method [13].

The DQMC method uses a discrete auxiliary field approach to formulate a computable form of the partition function [2], [4]. The auxiliary field is called the Hubbard-Stratonovich (HS) field, which is a collection of $NL$ numbers of value $+1$ or $-1$ to represent up spin and down spin for fermions. The simulation uses the Metropolis algorithm to find feasible spin configurations via local search on the HS field. Initially, a random configuration of the HS field is given. During the simulation, each element of the HS filed is visited, and a new configuration that flips the element's value is proposed. The acceptance of the new configuration is determined by the ratio of total energy of before and after flipping. A complete visiting of all $NL$ elements of the HS filed is called a *sweep*. Figure 1 uses pictorial representation of HS field to illustrate the flipping process.

The DQMC simulation consists of two stages: warmup stage and sampling stage. Each stage utilizes the Metropolis algorithm for different purposes. In the warmup stage, the Metropolis algorithm is used to thermalize the spin configurations; while in the sampling stage, it is employed to search the spin configurations that have a large Boltzmann factor of significant contribution to the partition function. The physical observables are then measured on the sampled spin configurations. Figure 2 displays the flow chart of the

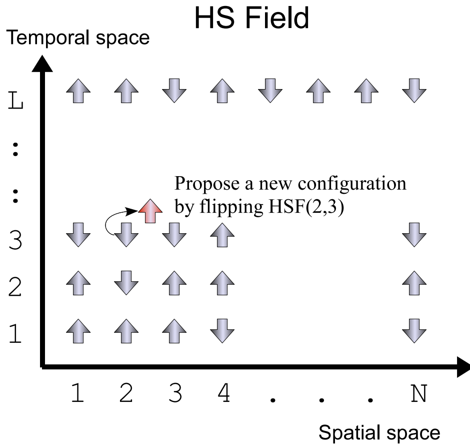


Fig. 1. Hubbard-Stratonovich (HS) field in the DQMC simulation.

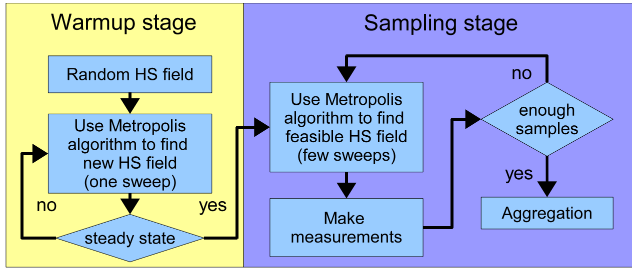


Fig. 2. Flow chart for the sequential DQMC simulation

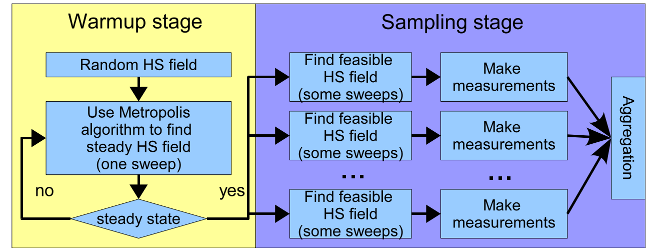


Fig. 3. Flow chart for the parallel Markov chain method for DQMC simulation.

sequential DQMC simulation.

The DQMC simulation for strong correlated materials face two major challenges, for which the parallelization techniques used in DCA++ are no longer applicable. First, the simulations for strongly correlated materials requires a much longer warmup stage to reach the steady state. This dramatically reduces the effect of the second level parallelization of DCA++. Second, the matrices in the DQMC formulation are more ill-conditioned. Therefore, the updating algorithms and the strategy of mixing single and double precision in DCA++ cannot sustain the desired accuracy for long. Frequent recalculation of whole matrices is essential to maintain the numerical stability.

In this paper, we focus on the parallelization techniques for the DQMC simulation of two-dimensional Hubbard model. A natural parallelization approach for the Monte Carlo simulation is to spread the physical measurements over parallel Markov chains, as shown in Figure 3. Each parallel chain uses a different initial random seed. This algorithm, named the parallel Markov chain (PMC) method, can effectively reduce the execution time of the sampling stage, but not that of the warmup stage. Therefore, the speedup is limited. There are several algorithms to parallelize the single Markov chain [6, Chap.16][7], [8] However most of these algorithms are designed for special problems [8] and generally is not scalable [7]. As pointed out in [6, Chap.16], it is necessary to incorperate other parallelization methods for efficient utilization of massively parallel processing units.

The fine-grained parallelization of the DQMC simulation also encounters some difficulties. The matrix size is usually too small to benefit by using standard parallel math packages, such as ScaLAPACK [9]. Multicore parallelization of basic matrix computations is more suitable [10], [11], [12]. But efficient implementation of some higher level matrix decompositions, such as the QR decomposition with column pivoting, are not fully supported in the current release of multicore math packages.

In this paper, we present a hybrid granularity parallelization (HGP) scheme that combines various algorithmic and implementation techniques to speedup the DQMC simulation. At the coarse-grained level, parallel Markov chain and task decompositions are carried out through MPI functions. At the fine-grained level, a new *rolling feeder algorithm* is designed to speedup the Green's function calculations. A parallel sparse matrix multiplication algorithm is introduced for multicore parallelization. Additional techniques for high performance computing, such as communication overlapping, message compression and load balance are also included in the HGP scheme. We show that with the challenges of the long warmup stage and ill-conditioned matrices, it is still possible to speedup the simulation by exploring the parallelism at different levels of granularity. We have observed 50x to 80x speedups of the HGP version of QUEST on an IBM Blue Gene/P parallel computer.

The rest of this paper is organized as follows. In section II, we briefly introduce the DQMC and computational kernels. A performance profile of a model problem is also presented. Section III illustrates three parallelization algorithms: parallel matrix computation, rolling feeder algorithm, and block matrix partition. Section IV describes the implementation techniques such as task decomposition, communication overlapping, message compression and load balance. Section V presents the performance results. Conclusion and future work are presented in section VI.

## II. DQMC SIMULATION

### A. Hubbard model

The Hubbard model is a fundamental model to study one of the core problems in material science: How do the interactions between electrons in a solid give rise properties like magnetism

and superconductivity. The Hubbard model for interacting fermions on a two-dimensional lattice is described by the Hamiltonian:

$$\mathcal{H} = \mathcal{H}_K + \mathcal{H}_V + \mathcal{H}_\mu,$$

where $\mathcal{H}_K$, $\mathcal{H}_V$, and $\mathcal{H}_\mu$ represent kinetic energy, potential energy and chemical energy, respectively. The expected value of a physical observable $\mathcal{O}$, such as density-density correlation, is given by

$$\langle \mathcal{O} \rangle = \mathrm{Tr}(\mathcal{O}\mathcal{P}),$$

where "Tr" is a trace over the Hilbert space describing all the possible occupation states of the lattice, and $\mathcal{P}$ is a distributed operator defined as

$$\mathcal{P} = \frac{1}{\mathcal{Z}} e^{-\beta\mathcal{H}},$$

and $\mathcal{Z}$ is the partition function, $\mathcal{Z} = \mathrm{Tr}(e^{-\beta\mathcal{H}})$, $\beta$ is the inverse of the product of the Boltzmann's constant $k_B$ and the temperature $T$, referred to as the *inverse temperature*,

$$\beta = \frac{1}{k_B T}. \tag{1}$$

By the path integral, the partition function is approximated by discretizing $\beta$ into $L$ intervals of length $\Delta\tau$,

$$\mathcal{Z} = \mathrm{Tr}(e^{-\beta\mathcal{H}}) = \mathrm{Tr}\left(\prod_{\ell=1}^{L} e^{-\Delta\tau\mathcal{H}_\ell}\right) + O(\Delta\tau^2). \tag{2}$$

Each $\mathcal{H}_\ell$ is then separated into the one body (kinetic) and two body (interaction) terms. The interaction term is further decoupled by the Hubbard-Stratonovich transformation, which uses an auxiliary filed, called Hubbard-Stratonovich (HS) filed, to reduce the problem to a quadratic form. This field depends upon both spatial site and time coordinate. For an simulation with $N$ site and $L$ time slice, the HS field contains $NL$ numbers, which can be either $+1$ or $-1$ in the discrete Hubbard model.

### B. DQMC simulation

The matrices in the Hubbard simulation, referred to as the Green's function, are of the form

$$G_\ell = (I + B_\ell B_{\ell-1} \cdots B_1 B_L \cdots B_{\ell+1})^{-1}. \tag{3}$$

The dimension of $G_\ell$ and $B_i$'s is $N$. Each $B_i$ is characterized by the parameters of the Hubbard model and the HS field at time slice $i$. The index $\ell$ of $G_\ell$ represents the index of the leftmost $B_i$. Matrices $B_i$ are multiplied in the circular order.

The DQMC simulation begins with a randomly generated HS configuration $\{h_{\ell,i}\} = \{\pm 1\}$. In each MC iteration, the simulation flips only one entry $h'_{\ell,i} = -h_{\ell,i}$ and leaves the rest unchanged. The acceptance of a new HS configuration is determined by the ratio

$$\rho = \frac{\det(\hat{G}_\ell)}{\det(G_\ell)},$$

where "det" is the determinant of a matrix, and $G_\ell$ and $\hat{G}_\ell$ represent the Green's function before and after the flipping. If $\rho$ is larger than a uniform random number $r \in [0, 1]$, the new configuration is accepted. Otherwise, the simulation keeps the current configuration and moves to the next configuration.

A complete loop of all HS configurations is called a *sweep*. In a typical simulation, thousands of sweeps at the warmup stage must be performed before the physical measurements at the sampling stage can take place. Similarly, thousands of sweeps need be performed to collect statistically meaningful physical measurements. The interested physical measurements fall into two major categories, equal-time and unequal-time. The time here refers to the *imaginary time* $\beta$, as defined in (1). The equal-time measurements are physical properties that are independent of $\beta$, such are density or energy. The unequal-time measurements are physical quantities that depends on $\beta$, such as susceptibility.

### C. Computational kernels

The Green's function defined in (3) is called the *equal-time Green's function*, which is one of the computational kernels in the DQMC simulation. It is a function of the HS fields. In one sweep, the number of proposed configurations is $NL$. Therefore, the same amount of $G_\ell$ need be calculated. The complexity of calculating $G_\ell$ is $O(N^3 L)$. Thus, the total computational complexity of completing one sweep is $O(N^4 L^2)$.

There are several ways to reduce the computational complexity [14]. Some of them are similar to those described in [1], such as the delayed updates. Here we present two techniques used only for the DQMC method.

1) The first one is to flip the HS configurations from the left most time slice. In this way, one can apply the technique presented in [1] for calculating the determinant and for updating Green's function. The complexity can be reduced to $O(N^3 L^2)$.
2) The second one is to calculate $G_{\ell+1}$ by using the relationship

$$G_{\ell+1} = B_\ell^{-1} G_\ell B_\ell. \tag{4}$$

This trick is called *wrapping*. It can quickly swap the next time slice to the left without recalculating the entire $G_{\ell+1}$. As a result, the computational complexity can be further reduced to $O(N^3 L)$ per sweep.

Those fast algorithms only work for small $\beta$ due to the numerical stability issue. When $\beta$ is large, as well as $L$, the product of $B_i$ has a large condition number. It is necessary to explicitly recalculate $G_\ell$ frequently. Thus the complexity is back to $O(N^3 L^2)$.

Another computational kernel is the calculation of the *unequal-time Green's function* $G^\tau_{k,\ell}$ for the unequal-time physical measurements:

$$G^\tau_{k,\ell} = \tag{5}$$
$$\begin{cases} (I + B_k \cdots B_1 B_L \cdots B_{k+1})^{-1} B_k \cdots B_{\ell+1} & k > \ell \\ (I + B_k \cdots B_1 B_L \cdots B_{k+1})^{-1} & k = \ell \\ -(I + B_k \cdots B_{k+1})^{-1} B_k \cdots B_1 B_L \cdots B_{\ell+1} & k < \ell \end{cases}$$

| Task | Percentage of time |
|---|---|
| Equal-time Green's function | 46.21% |
| MC iteration | 5.61% |
| Equal-time measurement | 1.96% |
| Unequal-time Green's function | 45.55% |
| Unequal-time measurement | 0.67% |
| Execution time (seconds) | 1,038.28 |

TABLE I
PROFILE OF A MODEL DQMC SIMULATION ON SINGLE NODE OF BLUE GENE/P.

Note that $G^{\tau}_{\ell,\ell}$ is the equal-time Green's function $G_\ell$. All $G^{\tau}_{k,\ell}$ share similar properties with the equal-time Green's function, including numerical instability and computational complexity of $O(N^3L^3)$. However, since the Green's function is periodic in the imaginary time, we only need to compute $G^{\tau}_{k,\ell}$ for a fixed $k$ or $\ell$ at the sampling stage.

### D. Performance profile

We had profiled the DQMC simulation using `gprof` on a single node of Blue Gene/P. The simulation uses $N = 256$ lattice sites, $L = 96$ time slices, $n_W = 1$ warmup sweep and $n_S = 20$ sampling sweeps.[1] The equal-time measurements are performed at every 12 wrapping, and the unequal-time measurements are sampled at every 5 sweeps. For the unequal-time measurements, only two rows and two columns of the unequal-time Green's function are calculated. The profiling result is given in Table I. Based on the profile, we can estimate that the execution time of a full simulation with $n_W = 1,000$ warm sweeps and $n_S = 20,000$ sampling sweeps would take about 288 hours (12 days) on a single node.

## III. PARALLELIZATION ALGORITHMS

The parallel Markov chain (PMC) method, as shown in Figure 3, has several advantages [6]. It is easy to implement and has a high computation-communication ratio. It is highly efficient for small number of parallel chains. However, it cannot be scaled well owing to the serial natural of the warmup stage. The limit of the PMC can be explained by Amdahl's law. Suppose that a sequential program with $n_W$ warmup sweeps and $n_S$ sampling sweeps. For $p$ processors, the parallel Markov chain method executes $n_W$ warmup sweeps and $n_S/p$ sampling sweeps on each processor, and then aggregates their results at the end. The speedup of the PMC method equals to

$$\rho_{\text{PMC}} = \frac{n_W + n_S}{n_W + n_S/p}. \tag{6}$$

It is easy to see $\rho_{\text{PMC}}$ is asymptotically bounded by $(n_W + n_S)/n_W$. For the profiling simulations in section II-D, $n_S/n_W = 20$, if $p = 10$, then the speedup is 7. However when $p = 400$, the speedup is only 20. The utilization of processing units drops rapidly as $p$ increases.

In the rest of this section, we introduce three new parallelization algorithms for better utilization of processors. They are parallel matrix multiplication, rolling feeder algorithm and block matrix partition. Those algorithms are independent of each other. It means that their speedups can be accumulated. The effectiveness of each algorithm will be demonstrated in section V.

### A. Parallel matrix multiplications

As shown in (3), the matrix-matrix multiplication is one of the numerical kernels in the Green's function calculation. Because of the small size of the matrices, standard parallel math libraries, such as PBLAS [9], do not provide a satisfactory solution to speedup the performance. Emerging multicore math libraries, such as FLAME [15] or PLASMA [16] might be more suitable. However, for the matrix size as small as a few hundreds, the speedup still cannot compensate the overhead of data partition and distirbution.

To speedup the matrix multiplication for calcuating Green's functions, we use the *checkerboard method* [14], [3]. By the *checkerboard method*, we first approximate the matrix $B_i$ by a product of four sparse matrices

$$B_i \approx C_4C_3C_2C_1.$$

Then let $A = (A_1\ A_2 \cdots A_N)$ be the column partition of the $N \times N$ matrix $A$ to be multiplied by $B_i$. With the sparse representation, $B_iA$ can be computed by $4N$ sparse matrix-vector multiplications:

$$B_iA = (B_iA_1\ B_iA_2 \cdots B_iA_N).$$

This method provides a natural data partition via the columns of $A$. Since each matrix-vector multiplication does not have dependency, the computation of $B_iA$ can be parallelized by a loop parallelization.

The multiplication $AB_i^{-1}$ can be calculated in a similar manner. To maximum the cache utilization, the multiplication is computed on the transposed form $B_i^{-T}A^T$, and the result is obtained by another matrix transpose.

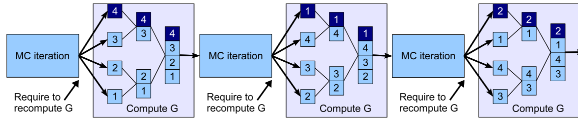
It should note that although the checkerboard method reduces the complexity from $O(N^3)$ to $O(N^2)$. However, due to the stabilization procedure, the overall complexity of Green's function calculation remains $O(N^3L)$.

### B. Rolling feeder algorithm

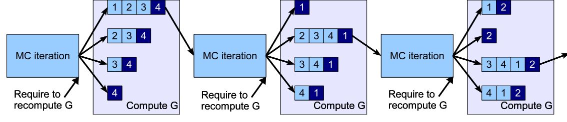
Due to the numerical stability issue, the equal-time Green's functions $G_\ell$ need be recalculated repeatedly, as mentioned in section II-C. Since the fast *wrapping* method (4) is used, the matrices $B_i$s in $G_\ell$ are multiplied in different order. Suppose the Green's functions need be fully recomputed after every $k$ wrappings. One can divide matrices $B_i$s into $m \equiv L/k$ groups:

$$\underbrace{B_1 \cdots B_k}_{A_1}\underbrace{B_{k+1} \cdots B_{2k}}_{A_2} \ \cdots\ \underbrace{B_{L-k+1} \cdots B_L}_{A_m}.$$

[1]The number of sweeps was scaled to 1/1,000 of the full simulation to reduce the profiling time.




(a) Parallel scan algorithm for multiplying a sequence of matrices.




(b) Rolling feeder algorithm for multiplying a sequence of matrices.

Fig. 4. Parallel scan algorithm and the rolling feeder algorithm

Then the recomputed Green's functions are

$$\begin{aligned} G_1 &= (I + A_1 A_2 \cdots A_m)^{-1}, \\ G_2 &= (I + A_2 A_3 \cdots A_1)^{-1}, \\ &\cdots \\ G_m &= (I + A_m A_1 \cdots A_{m-1})^{-1}. \end{aligned}$$

Note that the matrix $G_i$ cannot be computed until the Metropolis algorithm visit through the corresponding HS field in the time slice $i$ to get the the configurations of the left most matrix $B_i$.

The parallel scan [17, Chap.4] is commonly used to compute a long chain of matrix multiplication in parallel. It recursively multiplies two matrices, or two partial products in parallel. On $O(p)$ processors, the parallel scan can compute the product in $O(\log p)$ time. However, besides the obvious drawbacks of poor load balance, there are two reasons that the parallel scan method is not suitable for Green's function calculations. First, the cost of communication is high. It requires to pass the partial matrix products in the reduction step. Second, it is the numerical unstable to merge partial matrix products. In the sequential code, a stabilization process, referred to as the UDT decomposition, is performed for every $k$ matrix multiplications. However, the UDT decomposition cannot be used in the merging process. This is the main drawback.

Unlike the parallel scan, the proposed rolling feeder algorithm uses $O(p)$ processors and delivers the final matrix product in a constant time (independent of $p$). Let us call the processors that compute the matrix product the *feeders*, and the processor that uses the computed results the *iterator*. Each feeder accumulates the partial matrix products in different order until that all required matrices are multiplied.

Figure 4 uses an example to illustrate the parallel scan algorithm and the rolling feeder algorithm for computing the product of $m = 4$ matrices. The darker block in each "`Compute G`" box represents the $A_i$ block matrices with newly arrived HS field. In the beginning, the desired Green's function is $G_1 = (I + A_1A_2A_3A_4)^{-1}$, then the next one is $G_2 = (I + A_2A_3A_4A_1)^{-1}$, and the next one is $G_3 = (I + A_3A_4A_1A_2)^{-1}$. It should notice that the block matrices will continue to rotate, and only the left most block matrix is with the new HS field. Each time when it is required to recompute Green's function $G_i$, the parallel scan algorithm computes the matrix product from scratch. However, for the rolling feeder algorithm, there will be one of feeders that has completed the matrix product to feed to iterator. Other feeders continue the accumulation process.

| Task | Sequential | PS | RF |
|---|---|---|---|
| Matrix multiplication | $L$ | $\log L$ | 1 |
| Stabilization step | $O(L)$ | $O(\log L)$ | 1 |
| Inverting $(I + B_1 \dots B_L)$ | 1 | 1 | 1 |
| Data transmission | $N^2$ | $O(LN^2)$ | $N^2$ |

TABLE II
THE RESPONSE TIME TO GET $G_\ell$ OF THREE DIFFERENT ALGORITHMS.

Table II compares the response time of three algorithms: sequential, parallel scan (PS) and rolling feeder (RF). We treat the Green's function calculation as a black box with the HS field as input and the desired Green's function as the output. The response time is measured by the number of tasks need be completed from input to output. It can be seen that the time of getting $G_i$ by rolling feeder algorithm is independent of $L$. The data transmission of the rolling feeder algorithm is $N^2$ for sending the computed $G$ to the iterator.

### C. *Block matrix partition*

As shown in the profile of the DQMC in Table I, the unequal-time Green's function calculation takes almost half of the total time. A simple parallel method is to partition the block matrices into groups and use the equation (5) to compute them in parallel. There are two practical issues with this approach. The first one is the communication cost. Although the required information to the initial calculation is only the HS configurations of order $NL$, the data size of computed results is at the order of $N^2$. Section IV-C will present a compression scheme for the computed result. The second issue is the load balance. The different partitions of the group can significantly affect on the performance. Theoretical and empirical solutions will be presented in sections IV-D and V-B.

## IV. IMPLEMENTATION TECHNIQUES

In this section, we discuss the implementation of the HGP scheme for the parallelization of QUEST. We begin with an overview of the system design.

The parallel Markov chain algorithm described in section I evenly partitions the processing units into groups. Each group is called a *simulator*, as displayed in Figure 5. Two major components of the simulator is the *MC walker* and the *M-server*. A MC walker, consisting of one iterator and several feeders, employs the rolling feeder algorithm to carry out the MC iteration. A M-sever takes charge of physical measurements for both the equal- and unequal-time. It also contains the $G^\tau$-calculators for computing the unequal-time Green's functions. The arrows in the diagram represent the directions of message passing.

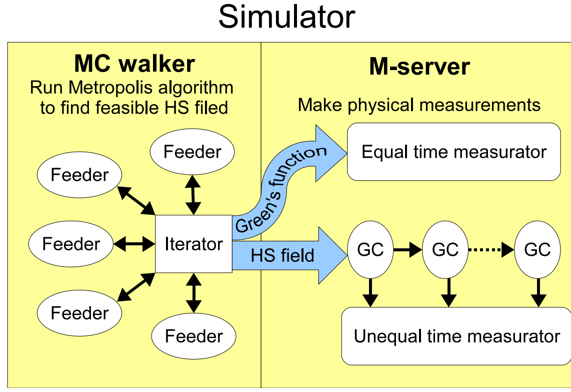
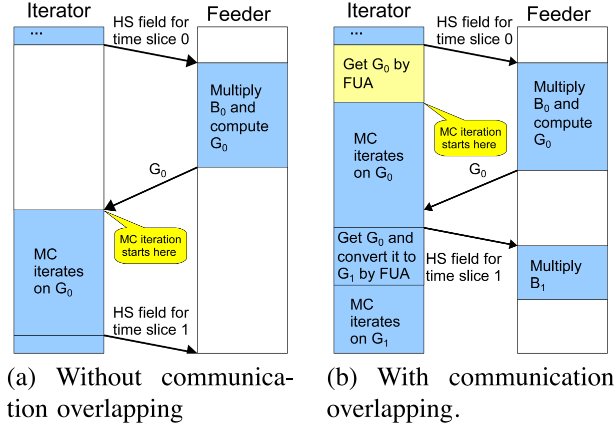



Fig. 5. Components of a simulator




(a) Without communication overlapping (b) With communication overlapping.

Fig. 6. The time diagrams for the rolling feeder algorithm without/with communication overlapping.

## A. Task decomposition

Let us examine the dependency of each component in a simulator and consider how to parallelize independent tasks. Two tasks at the top level are the MC iteration and the measurements. The MC iteration does not depend on the physical measurements. On the other hand, the physical measurements require the HS fields generated by the MC iteration at the beginning of the computation. Thus the execution of the MC iteration and physical measurements can be overlapped.

The dependency of the feeders and the iterator are explained in section III-B. They are strongly coupled, but most of computations can be overlapped.

There are three types of components inside the M-server: $G^\tau$-calculator (GC), equal-time measurator, and unequal-time measurator. In the calculation of the unequal-time Green's function, $G^\tau_{k,\ell}$ can be computed in parallel by using the equation (5). Therefore, $G^\tau$-calculators can be executed in parallel. Although some intermediate results of them can be shared, different physical measurements are independent of each other in general. Hence parallelization on different measurators can be carried out easily.

## B. Communication overlapping

In the rolling feeder algorithm, there is data dependency between the feeders and the iterator. As shown in Figure 6(a), the feeder needs the HS field to construct $B_0$ and then to compute $G_0$. The iterator waits for the feeder to send the $G_0$ back to continue the simulation. Although the response time is constant (independent of $L$) as analyzed in section III-B, we can use the Fast Update Algorithm (FUA) as described in (4) to overlap this communication and speedup the response time.

Figure 6(b) shows the use of the FUA to speedup the response time. When the iterator sends out the HS field for computing $B_0$, instead of waiting for $G_0$, it computes $G_0$ using the FUA locally. After that, the MC iteration on $G_0$ can start immediately, using the local computed $G_0$. After receiving the HS field, the feeder uses it to compute $B_0$ and $G_0$ and then sends back to the iterator. Note that the newly computed $G_0$, received from the feeder, is better than the locally computed one in the sense that it contains more significant digits. Therefore, the iterator will use the received one to continue the iteration.

| Without comm overlapping | With comm overlapping |
|---|---|
| trans time for HS field<br>one matrix multiplication<br>one stabilization step<br>one matrix inversion<br>trans time for $G$ | two matrix multiplication |

TABLE III
THE TASKS NEED BE FINISHED TO GET $G_0$ FOR METHODS WITHOUT/WITH COMMUNICATION OVERLAPPING.

It can be seen that the response time for the iterator to start the MC iteration in Figure 6(b) is much shorter than that in Figure 6(a). The quantitative comparison for those two methods can be measured by the tasks to be finished for getting $G_0$ shown in Table III. The one with communication overlapping uses additional computation to reduce the waiting time for getting $G_0$ from the feeder. One may question the accuracy of the local computed $G_0$, since the result produced by the FUA is not as stable as the recomputed one. However, we can control it at the beginning of system design. Suppose the FUA can be applied only $k$ times to compute $G_\ell$. The frequency of recomputing $G_\ell$ can be increased to every $k-1$ updates, instead of $k$. This change can guarantee the stability of the FUA.

## C. Message compression

There are three kinds of messages inside a simulator: HS field, the equal-time Green's function and the unequal-time Green's function. HS field is sent by the iterator to feeders and $G^\tau$-calculators. Since feeders need partial HS field to build Green's function, the frequency of message passing between iterator and feeders is high. To reduce the message size, HS field is compressed into the bitmap format. The message size is reduced a factor of 16.

Equal-time Green's functions are transmitted from feeders to iterator and measurator. Although it is not passed as frequently as the HS field, each message has much larger

size. A simple compression scheme is to convert the precision from double to single. The conversion is a safe for numerical stability since the high accuracy is only required for computing Green's function, not for the use of it. A factor of 2 compression ratio is gained via this simple approach. More sophisticated compression schemes are abandoned since they increase the processing time.

Unequal-time Green's functions are sent from $G^\tau$-calculators to the unequal-time measurator. Although the frequency of transmission is low, the size of messages is very large. Besides the type conversion from double to single, the translational invariance properties of the elements of the unequal-time Green's function are also used to compress the message. This can reduce the message size from $O(N^2)$ to $O(N)$.

### D. Load balance

After all aforementioned techniques are applied, the iterator becomes the performance bottleneck. In fact, this is the best scenario that one can get since the MC iteration cannot be parallelized at all.

Here we consider the issue about the utilization of processors. If $G^\tau$-calculators are idle most of the time, we can lump the computation of two or more $G^\tau$-calculators into one. By doing so, the message amount can be also reduced.

The simple queueing theory can help the estimation of how many $G^\tau$-calculators should be merged. Let $\lambda$ be the average arrival rate of new HS field (or equivalently, $1/\lambda$ is the average time that iterator generates a new HS field). Let $T$ be the average time to process a new HS field for one $G^\tau$-calculator and $n_C$ be the number of $G^\tau$-calculators to be lumped. If we lump $n_C$ $G^\tau$-calculators into one processor, the average time to process a new HS field is $n_C T$. By Little's law [18], the processor utilization $P$ can be estimated as

$$P = \lambda n_C T.$$

Therefore, for the best processor utilization,

$$n_C = \max_{P \leq 1} \left\lfloor \frac{P}{\lambda T} \right\rfloor \leq \left\lfloor \frac{1}{\lambda T} \right\rfloor. \quad (7)$$

We will see the empirical results for how this estimation works in section V-C.

## V. Performance results

We implemented the HGP scheme with MPI and OpenMP for the parallelization of QUEST [13] on modern heterogeneous high performance computing. In this section, we present performance results on an IBM Blue Gene/P. The Blue Gene/P uses the system-on-chip technology to have small and lower-power chips connected through five specialized networks. Each compute node is equipped with 850MHz PowerPC 450 quad-core processor and 2GB memory. We use IBM XL compilers with BLAS and LAPACK libraries.

Three experiment results are presented in this section. The first one is on the communication pattern of the different components inside simulators. The second one concerns the performance of the HGP scheme with respect to different number of time slices $L$. The third one is about the lumping method proposed in section IV-D.

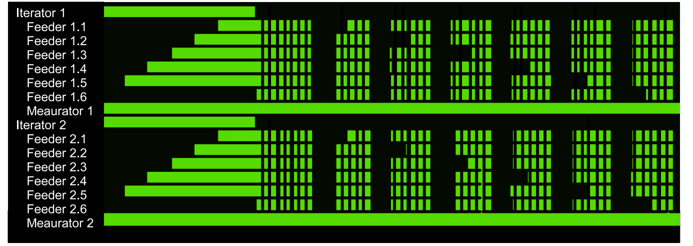



Fig. 7. Communication patterns for iterator, feeders, and measurator.

### A. Communication patterns

The communication patterns for two simulators are displayed in Figure 7. Each simulator consists of one iterator, six feeders, and one equal-time measurator. The horizontal strips represent the communication pattern of computational components over time. The bright regions in the strip represent the waiting time caused by `MPI_RECV`; the dark regions mean some other activities other than MPI communication.

The rolling feeder algorithm runs like the pipeline method plus the master-slave model. As shown in Figure 7, at the beginning, the 6th feeder computes the first Green's function $G_L$ and sends it to the iterator. Other feeders compute the partial products of $B_i$s to different length, according to their assignments. Once the iterator gets the Green's function, it iterates through all sites, and sends out the new HS configurations to feeders. The feeders multiply more $B_i$s based on the received HS fields. After 8 rounds, the first feeder computes $G_8$ and sends it to the iterator. Meanwhile, other feeders perform the matrix multiplication and stabilization step for the partial product. The same pattern repeats when the subsequent feeders deliver $G_\ell$ to the iterator in turn.

Figure 7 also shows the effectiveness of the communication-computation overlapping technique. There is no bright regions for the iterator after it gets the first $G_\ell$. Note that the measurators are idle to wait for the first feasible HS field since the simulation starts from the warmup stage.

### B. Speedup for different $L$

This experiment is to show the speedups of the DQMC simulation on a two-dimensional periodic lattice. The lattice size is $N = 16 \times 16 = 256$, and the number of time slices is varied from $L = 24, 48, 96$ to 192. The number of sweeps are 10 and 200 for the warmup and sampling stages, respectively.[2]

The parameters for the kinetic energy, potential energy and chemical potential of the Hubbard model are set as $t = 1$, $U = 4$, and $\mu = 0$ (half-filling), respectively. Due to large potential energy $U$, the equal-time Green's function needs be

[2]We scaled down the simulation proportionally to reduce the total simulation time.

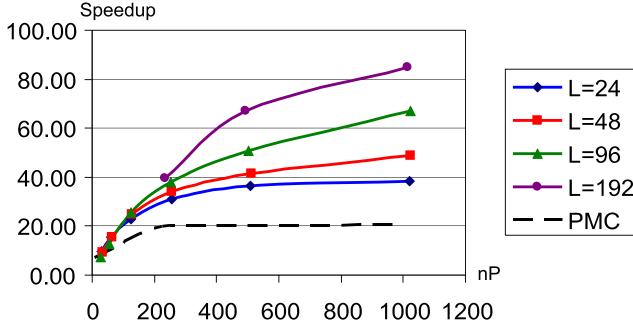



Fig. 8. Speedup for various $L$ with equal time measurements only.

recomputed after every 8 wrappings. The physical measurements include density, energy, magnetic structure factors, spin-density correlations, and superconducting correlations [4]. The equal-time measurements are made for every 12 wrappings.

Figure 8 displays the speedups on the different number of processors. It can be seen that the larger $L$ is the better speedup is. The reason is that there is a high percentage of time spent on the calculation of the unequal-time Green's function for large $L$. Thus, the HGP scheme is more effective. Comparing to the speedup of using the parallel Markov chain (PMC) method only, shown by the dashed line in Figure 8, the HGP scheme demonstrates significant performance improvement.

We can use Amdahl's law to explain the performance results. The speedup of the parallel Markov chain (PMC) is given in equation (6). The same technique can be used to estimate the speedup of the rolling feeder algorithms, for which the parallelizable part is the matrix multiplication and the sequential part is the matrix inversion.

The speedup $\rho_{\mathrm{TD}}$ of the task decomposition is determined as follows:

$$\rho_{\mathrm{TD}} = \frac{T_G + T_I + T_P}{\max\{T_G/\rho_{\mathrm{RF}}, T_I, T_P\} + c_1}, \quad (8)$$

where $T_G$, $T_I$, and $T_P$ are the time for equal-time Green's function calculation, MC iteration, and physical measurements, respectively; $\rho_{\mathrm{RF}}$ is the speedup by the rolling feeder algorithm and the parallel matrix computation; parameter $c_1$ is the overhead. The performance model of our HGP scheme is then given by the product of the speedup of parallel Markov chain and the speedup of task decomposition:

$$\rho = \rho_{\mathrm{PMC}}\rho_{\mathrm{TD}} \quad (9)$$

Table IV lists the optimal theoretical speedups and the practical ones. The theoretical results are obtained based on the profiling results and assuming $c_1 = 0$. The column "nP" is for the number of processors; Column "Empirical speedup" records the actual speedup obtained from experiment results.

### C. Speedup for different lumping factor $n_C$

Our third experiment is to test the lumping method presented in section IV-D. It uses the same setting as that in the second experiment, except fixing $L = 96$. In addition, the unequal-time measurements are performed in every 5 sweeps. At every measurement point, two block rows and two block columns of the unequal-time Green's function are calculated. Therefore, there are $4L = 384$ submatrices. The block matrix partition algorithm parallelizes the computations by distributing the submatrices to $p$ processors. Section IV-D gives an expression to determine the optimal number of processor $p$ to use.

| L | nP | $\rho_{\mathrm{PMC}}$ | $\rho_{\mathrm{TD}}$ | Theoretical ($\rho_{\mathrm{PMC}}\rho_{\mathrm{TD}}$) | Empirical speedup |
|---|---|---|---|---|---|
| 48 | 512 | 16.0 | 3.8 | 60.83 | 41.56 |
| 48 | 1024 | 18.2 | 3.8 | 68.11 | 48.79 |
| 96 | 504 | 13.5 | 5.2 | 69.83 | 50.71 |
| 96 | 1022 | 16.4 | 5.2 | 85.26 | 66.97 |
| 192 | 494 | 10.2 | 6.8 | 69.56 | 67.01 |
| 192 | 1014 | 13.9 | 6.8 | 94.38 | 84.95 |

TABLE IV
THEORETICAL SPEEDUP AND THE EXPERIMENT RESULT.

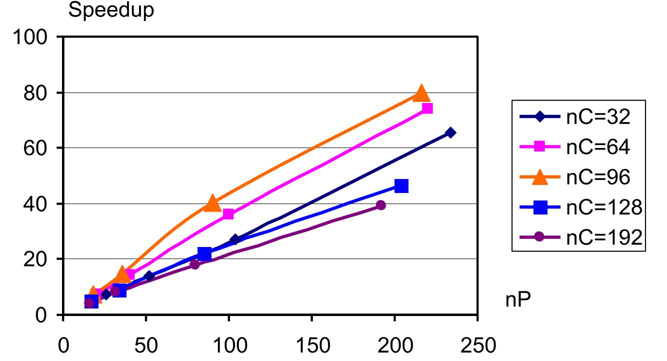



Fig. 9. Speedup for different lumping parameter $n_C$.

The lumping factor $n_C$ is the number of submatrices needed be computed on a single processor. In this experiment, the lumping factor $n_C = 4L/p$. Figure 9 presents the speedups for different lumping factor. As we can see, when $n_C$ increases from 32 to 96, the performance is increased accordingly. However, as $n_C$ continuously increases from 96 to 128 and 192, the performance actually drops. A simple explanation is that when $n_C \leq 96$, the bottleneck is the MC iteration. Therefore, with less processors per simulator, the system can have more simulators to speedup the performance. However, once the lumping factor $n_C$ is larger than 96, the physical measurements become the bottleneck and the overall performance drops.

We can actually use the equation (7) to estimate the optimal lumping factor. From the profiling data, the time to compute a single block matrix is about $T = 0.152$ second. Suppose after the parallelization, the iterator is the bottleneck of performance. Therefore, the average time to finish a sweep can be estimated through dividing the total running time by the total number of sweeps. In the current setting, it is 3.28 second. Thus, the arrival rate of request is $\lambda = 1/(5 * 3.28) = 0.06$, where the number 5 is how often the unequal-time mea-

surements are performed in terms of the number of sweeps. As given in the equation (7), the optimal lumping factor is $n_C^* = 1/(\lambda T) = 108$. Thus, for $n_C \leq n_C^*$, the $G^\tau$ calculator is not the bottleneck of the simulation. Once $n_C > n_C^*$, the running time of simulation is determined by the execution of $G^\tau$-calculator. Therefore, the longer execution time of $G^\tau$-calculator, the worse performance results.

## VI. Concluding Remarks

The DQMC simulation has been studied for nearly three decades. However, challenges of implementing DQMC simulations involving Markov chain, small matrix dimensionality and numerical stability on massively parallel machines have not been well studied. In this work, we have tried to address these challenges on the modern heterogeneous high-performance computing systems in a comprehensive way. We proposed a number of approaches to explore the parallelism at different levels. We implemented the proposed approaches to verify their effectiveness. The performance result achieves 80x speedup in the total simulation execution time on an IBM Blue Gene/P systems.

Simulations on larger lattice to bridge the gap between different models and approaches are highly desired. The study on three-dimensional lattices is a subject of future study. The parallel schemes that can reduce the cost of warmup stage are under investigation. Multi-band model of consisting of long range hopping is highly desired. Parallelization of continuous time QMC method would be of our great interests.

It is expected that the number of cores per processor will continuously increase. Therefore, fine-grained parallelization is inevitably to be a vital way that enhance the performance. We are also anticipate the availability of mature and scalable algorithms and software for multicore matrix computations.

The needs for efficient memory and I/O managements would become a challenge as more complicated physical measurements and statistical methods are included in the simulations.

## Acknowledgment

We thank Richard Scalettar for his guidance and stimulating discussions on the DQMC simulation. This project is supported in part by the Office of Science of the U.S. Department of Energy under SciDAC (Scientific Discovery through Advanced Computing) grant DE-FC-02-06ER25793. An early parallel implementation of the QUEST used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

The first author would like to acknowledge National Science Council of Taiwan for the support under the grant NSC98-2218-E-007-006-MY3, and National Center for High-performance Computing for using the computing and facilities.

## References

[1] G. Alvarez, M. S. Summers, D. E. Maxwell, M. Eisenbach, J. S. Meredith, J. M. Larkin, J. Levesque, T. A. Maier, P. R. C. Kent, E. F. D'Azevedo, and T. C. Schulthess, *New algorithm to enable 400+TFlop/s sustained performance in simulations of disorder effects in high-Tc superconductors*, in Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, pp.1–10, 2008.

[2] R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, *Monte Carlo calculations of coupled Boson-fermion systems*, Phys. Rev. D 24, vol.24, pp.2278–2286, 1981.

[3] E. L. Loh Jr. and J. Gubernatis, *Stable numerical simulations of models of interacting electrons in Condensed Matter Physics*. In Electronic Phase Transition ed. by W. Hanke and Yu. V. Kapeav, Elsvier Science Publishers, 1992.

[4] R. R. dos Santos, *Introduction to quantum Monte Carlo simulations for fermionic systems*, Brazilian J. of Phy., Vol.33, pp.36–53,2003.

[5] T. Maier, M. Jarrell, T. Pruschke, and M. Hettler, *Quantum cluster theories*, Reviews of Modern Physics, vol.77, pp.1027–1080, 2005.

[6] E. J. Kontoghiorghes, Ed., *Handbook of Parallel Computing and Statistics*. Chapman and Hall, 2006.

[7] A. E. Brockwell., *Parallel Markov chain Monte Carlo simulation by pre-fetching*, J. of Computational and Graphical Statistics, vol.15, pp.246–261, 2006.

[8] R. Ren and G.Orkoulas, *Parallel Markov chain Monte Carlo simulations*, J. Chem. Phys, vol.126, 211102, 2007.

[9] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. SIAM Press, 1997.

[10] G. Quintana-Orti, E. S. Quintana-Orti, E. Chan, F. G. van Zee, and R. A. van de Geijn, *Programming algorithms-by-blocks for matrix computations on multithreaded architectures*, FLAME Working Note, Tech. Rep. 29, Department of Computer Science, University of Texas, Austin, 2008.

[11] A. Buttari, J. Langou, J. Kurzak, and J. J. Dongarra, *A class of parallel tiled linear algebra algorithms for multicore architectures*, LAPACK Working Note 191, 2007. Available online: http://www.netlib.org/lapack/lawnspdf/lawn191.pdf

[12] V. Volkov and J. W. Demmel, *LU, QR and Cholesky factorizations using vector capabilities of GPUs*, LAPACK Working Note 202, 2008. Available online: http://www.netlib.org/lapack/lawnspdf/lawn202.pdf

[13] C-R. Lee *et al*, *QUEST: Quantum electron simulation toolbox*, http://www.cs.ucdavis.edu/∼bai/QUEST_public, 2009

[14] Z. Bai, W. Chen, R. Scalettar, and I. Yamazaki, *Numerical Methods for Quantum Monte Carlo Simulations of the Hubbard Model*, in "Multi-Scale Phenomena in Complex Fluids" ed. by T. Y. Hou, C. Liu and J.-G. Liu, Higher Education Press, pp.1–110, 2009.

[15] P. Bientinesi, E. S. Quintana-Orti, and R. van de Geijn, *Representing linear algebra algorithms in code: The FLAME APIs*, ACM Transactions on Mathematical Software, vol. 31, pp. 27–59, 2005.

[16] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, S. Tomov, *umerical linear algebra on emerging architectures: The PLASMA and MAGMA projects*, Journal of Physics: Conference Series, Vol. 180, 2009.

[17] T. G. Mattson, B. A. Sanders, and B. L. Massingill, *Patterns for Parallel Programming (Software Patterns Series)*. Addison-Wesley, 2004.

[18] J. D. C. Little, *A proof of the queueing formula $l = \lambda w$*, Operations Research, vol. 9, pp.383–387, 1961.