Development Of A Model For Scaling Quantum Monte Carlo Beyond Millions Of Cores

Ping-Ko Chiu*
Department of Computer Science, University of Illinois at Urbana-Champaign

Lucas K. Wagner†
Department of Physics, University of Illinois at Urbana-Champaign

INTRODUCTION

Simulations of quantum systems are critical to understanding materials. These simulations represents a large portion of supercomputing time. One of the most accurate methods to simulate realistic models of materials is quantum Monte Carlo (QMC)[1, 2]; however, QMC techniques are very expensive computationally which limits their applicability. Current QMC calculations are performed using massive parallelization; these calculations have used on order of 1 million threads concurrently.

In QMC, a Markov-Chains process, walker, is utilized to sample the wavefunction of the system. QMC methods can be parallelized by utilizing one walker per core with high parallel efficiency. This corresponds to very efficient scaling for sampling more complex systems.

The warm-up time is a limiting factor for very large scale parallelism. The QMC sampling process for each walker is composed to two parts: warm-up and production. Warm-up time is inherent to the system and is required for the sampling process to begin generating statistically significant data. Production time is after warm-up and is where the algorithm generates independent points for sampling the wave function. To reduce production time, more computer cores may be used for additional walkers. Each additional walker increases the data generation rate which reduces the total production time. However, the warm-up time of each walker is fixed. This makes it a bottleneck for large scale parallelism.

As computer architectures become more parallel, the number of cores available at super computing sites can be expected to reach the millions. We would like to design our algorithm so that we can fully utilize the extra parallelism. The time reduction for the QMC sampling process can be used to further accelerate the search for materials with designer properties. However, with the current design, the warm-up time for individual walkers begins to dominate the cost of the calculation when more walkers are utilized. Only performing paralellization on the walker level is very limiting. To obtain higher scalability, one needs to use multiple cores to accelerate a single walker and to reduce the warm-up time. There are several ways to do this. One way is to tackle fine-grained parallelism by targeting portions of the code that can be executed independently. In QMC calculations, the number of operations to perform one step for one electron in one walker is quite small, even for large simulations. It is thus challenging to parallelize the walkers on the step level of computation. The other way is to use algorithms that can improve sampling efficiency and be trivially parallelized. Under these algorithms, each walker can have greater sampling efficiency at the expense of greater computational resources.

In this article, we investigate an alternative way of achieving fine-grained parallelism in quantum Monte Carlo by adopting sampling algorithms that can use parallel processing. We apply a multiple try Metropolis algorithm[7] and assess its performance on realistic systems. We also compare it against the performance of the Metropolis Hastings algorithm[3] and the Splitting/Delayed Rejection algorithm[4, 5]. The sampling algorithms are implemented for the QWalk program where the performance is measured. QWalk[6] performs high accuracy QMC calculations of electronic structures. We then construct a model to estimate the viability of this approach for QMC calculations up to 10^9 cores. We found that by using multiple cores per walker, we can extend the parallelism of QMC calculations.

METHODS

Sampling algorithms

Metropolis Hastings Algorithm The Metropolis Hastings Algorithm by Hastings et al.[3] samples a complex distribution by proposing a single move for each walker, $\mathbf{R}$. The Metropolis-Hastings Algorithm suffers from high variance in estimates, and has a trade-off between convergence and acceptance that cannot be improved[7]. The Metropolis-Hastings algorithm[3] is also sequential by design. Therefore the algorithm is not easily parallelizable.

Splitting/Delayed Rejection Algorithm Mira[4] and Bressani et al.[5] proposed the Splitting/Delayed Rejection Algorithm that improves on the Metropolis Hasting algorithm by modifying the rejection criteria. When a proposed move is rejected, another move that has a higher acceptance probability is proposed. This recursive retry process is attempted until a specified number of retries(Depth) is reached. Thus, it allows for the sampling of a larger space at each step. Mira showed that the Splitting Rejection Algorithm produces smaller asymptotic variances for estimates when compared to the Metropolis-Hastings algorithm under fixed CPU time[4]. Since the retry process is independent of each other, each retry may be computed in parallel and be used only one the previous try has failed.

Multiple-try Metropolis Algorithm Liu et al.[8] proposed the Multi-try Metropolis Algorithm(MTM) that selects a single move from a set of trial moves in each step based on a probability distribution. Pandofi el al.[7] proposed the General Multi-try Metropolis Algorithm(GMTM) where the selection probability is relaxed to allow efficient calculations of probabilities. MTM Algorithms allow for sampling of larger spaces and acceptance can be improved by increasing the number of trial moves without sacrificing convergence. Since each try in the MTM algorithms are independent in nature, we can calculate each try in parallel and join the results for a selection at the end of each step.

The following is the GMTM pseudo code detailed by Pandolfi et al.[7] modified for the context of QMC calculations.

```
1. Calculate wave function data for current point $R_n$
2. Generate $k$ trials(denoted by $\rightarrow$), for each trial $i$
    (a) For each dimension generate forward translation according to
        $R_{n+1} = R_n + \chi\sqrt{\tau} + V_D(R_n)$ where $\chi \sim N(0,1)$
    (b) Calculate wave function $\Psi(i)$ for new point
    (c) Calculate $\ln(w_{(\rightarrow,i)})$ according to the weighting scheme
        • $MTM_I$ Scheme
          $\ln(w_{(\rightarrow,i)}) = 2*\ln|\Psi(i)|^2 - \frac{\Sigma_{d\in Dim}(-R_{n+1} - V_D(R_{n+1}))^2}{2*\tau}$
        • $MTM_{Inv}$ Scheme
          $\ln(w_{(\rightarrow,i)}) = 2*\ln|\Psi(i)|^2 + \frac{\Sigma_{d\in Dim}(R_{n+1} - V_D R)^2}{2*\tau}$
3. Choose trial based on probability $p_y$ of each trial
        $\frac{w_{(\rightarrow,i)}}{\Sigma_{i\in k} w_{(\rightarrow,i)}}$
   Let chosen index be denoted by $c$
4. Repeat Step 2 for $k$ realizations(denoted by $\leftarrow$), fixing one realization as $R_n$
5. Compute Acceptance Criterion by
        $a = \frac{w_{(\rightarrow,c)} p_{(\leftarrow,R_n)}}{w_{(\leftarrow,R_n)} p_{(\rightarrow,c)}}$
```

Term	Identifier
Metropolis Hastings Algorithm	M
Split Algorithm with k-Retries	Sk
GMTM with MTM_I weighting and k-trials	GI_k
GMTM with MTM_{INV} weighting and k-trials	$GInv_k$

TABLE I: In order to test the performance of the various algorithms, the algorithms have been implemented in QWalk. We gave the algorithms identifiers for simpler references later on in the article.

A force bias term, η has been incorporated into all algorithms to improve the acceptance ratio. The choice of η should be one that moves the walker, $\mathbf{R}$, in the direction of increasing $|\mathbf{\Psi}(\boldsymbol{R})|$[9]. Umrigar et al.[10] has proposed limits to η that would prevent the drift from diverging near nodes for Green's Function Monte Carlo. This limit in drift has also been shown useful for Variational Monte Carlo in QWalk. The acceptance criterion has to be adjusted to allow the biased moves.

We investigate the performance of the three algorithms on $Ca_2CuO_2Cl_2$, calcium oxychloride, a material that has been recently studied using QMC. It contains transition metals which are computationally challenging and has more correlation in the MCMC process compared to simpler systems. The high correlation makes it more difficult to extract independent points from the product phase.

Performance metrics

Performance will be measured by the acceptance ratio, independent points per step, and sampling efficiency.

The acceptance ratio indicates the proportion of walker steps that are accepted. It is a fine-grained estimate of algorithm performance that does not consider the computation cost of each step. Since we are introducing new algorithms for the QMC process, we need to ensure that the sampling has to be improved. The sampling performance is indicated by the acceptance ratio. The multiple try Metropolis algorithm and the splitting algorithm are designed to utilize multiple tries to enlarge the sampling space per step. We expect the acceptance ratio to be higher than that of single proposal algorithms.

The independent points per step measures how efficient each step is at sampling the distribution. A value of zero indicates that the sampling process could not generate any statistically significant data and a value of one represents perfect sampling. Compared to the acceptance ratio, the independent points per step gives a more holistic estimate of the sampling efficiency. However, this measure still does not consider the computational costs of each step.

Sampling efficiency takes into account the computational costs of the different algorithms. It is the ratio of the independent points per step of algorithms that use k-cores to the maximum independent points per k steps of the Metropolis algorithm. This gives an estimate of the percentage efficiency achieved at each step under fixed computational resources. Sampling efficiency for S_k algorithms assume the computation of all k retries. That may not be the case if the number of cores is less than k and some retries are omitted because the previous try is accepted. Thus the actual sampling efficiency for S_k could be higher depending on the number of cores allocated for the S_k algorithm.

Performance model

We built a model to predict performance of the algorithms for larger systems. The model should be able to capture the behavior of the algorithm given their sampling efficiency We measure performance in terms of the time of computation for a system like calcium oxychloride. The model takes into account the number of total steps, N_{steps}, the number of total cores, N_{core}, the number of independent points, N_{ind}, warm-up time in units of steps per independent point, t_w, and the decorrelation time in units of steps per independent point, t_d. Under constant number of total cores, the number of total walkers, N_w depends on the algorithm of choice.

$$N_w = \frac{N_{core}}{N_{core/walker}}$$

For an algorithm like the GI_4 the number of walkers would be a quarter of the number of walkers for M.

The time it takes for one walker to take one step depends on the computation sampling efficiency of the algorithm, ϵ, the number of cores per walker, $N_{core/walker}$, and the time per step of the Metropolis Hastings algorithm, T_{1core}.

$$T_{1w1s} = \frac{T_{1core}}{\epsilon N_{core/walker}}$$

The wall time and the CPU time of computation is then

$$\begin{aligned}
T_{wall} &= T_{1w1s} N_{steps} \\
&= T_{1w1s}\left(t_w + t_d \frac{N_{ind}}{N_w}\right) \\
&= T_{1w1s} t_d \left(\frac{t_w}{t_d} + \frac{N_{ind}}{N_w}\right) \\
&= \frac{T_{1core} t_d}{\epsilon N_{core/walker}}\left(\frac{t_w}{t_d} + \frac{N_{ind} N_{core/walker}}{N_{core}}\right) \\
T_{CPU} &= T_{wall} N_{core}
\end{aligned}$$

We have chosen a few constants for the purpose of modeling. These constants can be adjusted for different problems. We assume that the warm up time and the decorrelation time differ by a constant factor 1, $\frac{t_w}{t_d} = 1$, the one walker one step time $T_{1w1s} = \frac{1}{t_d}$ so $T_{1w1s} t_d = 1$, and $N_{ind} = 10 * 7$.

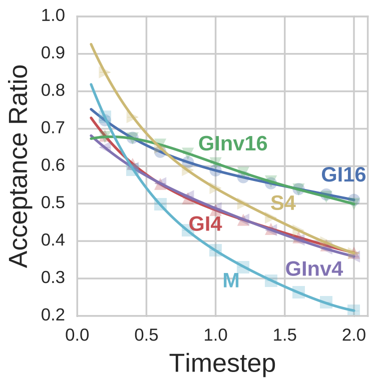

FIG. 1: Acceptance ratio of all algorithms

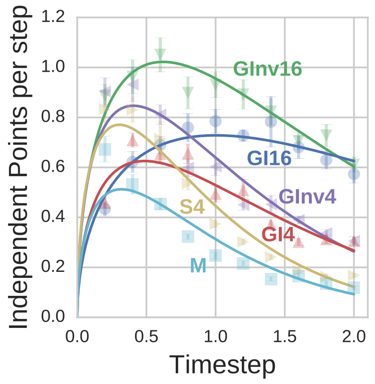

FIG. 2: Independent points per step of all algorithms

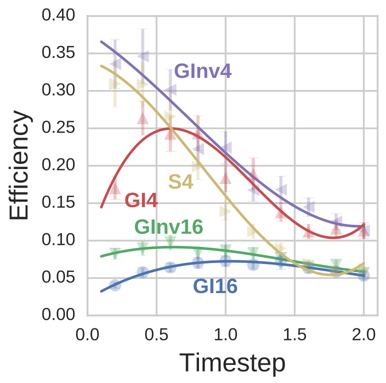

FIG. 3: Efficiency of algorithms as a ratio of the number of independent points to that of the Metropolis Hastings algorithm

RESULTS

Higher number of trials resulted in higher acceptance ratio. FIG. 1 shows the acceptance ratio of each algorithm. The data points are fitted with linear least squares of 4th order polynomial. As the number of trials for the GI, $GInv$, and S algorithms increase, the acceptance ratio also increased. The acceptance ratio of multi-try algorithms is also higher than that of single try algorithms. Since the multi-try algorithms are able to sample a much larger space per step, their acceptance ratio is higher under a fixed time step. This result shows that multi-try methods can improve the sampling process over single-try methods.

Besides the improved acceptance ratio, GI and $GInv$ showed a slower decay of acceptance ratio as step size increases. Compared to M and S algorithms, GI and $GInv$ has a slower decay at high time steps. This could be useful to ensure good performance even in non-optimal conditions. It could also save the optimization costs for finding the optimal time step.

The two sampling schemes of $GMTM$, MTM_I and MTM_{Inv}, showed no distinguishable difference in terms of acceptance ratio. Both performed very similarly at all time steps.

Multi-try algorithms like GI, $GInv$ and S showed higher independent points per step than M. FIG. 2 shows the independent points per step at various time steps. The data is fitted with Linear Least Squares of $y = At^{0.5}e^{-Bt}$ where A, B are constants, y intercept at 0. The 0.5 constant is chosen as an estimate of the effect of the diffusion process. Since multi-try algorithms are able to increase acceptance ratio by increasing the number of trials, the independent points per step also increases accordingly. We observe again that the GI and $GInv$ have slower decay at high time steps.

However, multi-try algorithms are less efficient generating of independent points. Even though multi-try algorithms exhibit higher acceptances and independent points per step, they are less efficient considering the computational resources that are required. The 4-try methods have a maximum efficiency around $0.25 - 0.35$ while the 16-try methods have maximum efficiency around $0.05 - 0.10$.

Model Results

The model allows us to estimate the computational time of any algorithm given its efficiency and number of cores per walker. FIG. 4 shows the charged(CPU) time vs. wall time of different strategies. "Standard" represents the strategy of 1-core at 100% efficiency. "$nc\epsilon$%" represents the strategy of n-cores at ϵ efficiency.

We see that each strategy approaches asymptotes in both axis. As the charged time approaches the horizon-

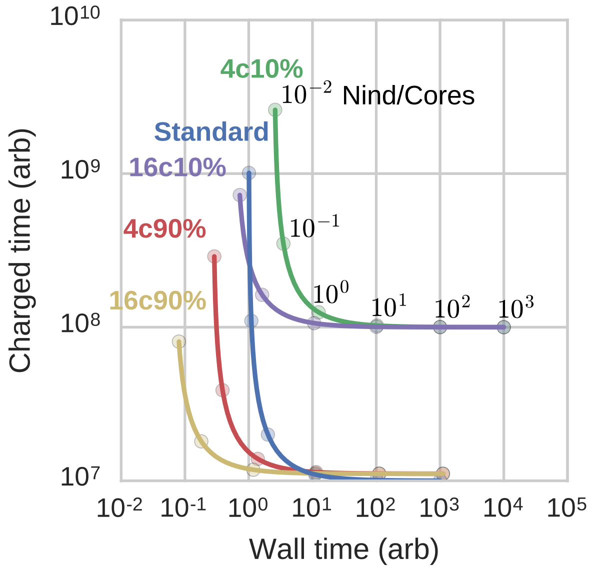

FIG. 4: Charged time vs. Wall time of various strategies at different number of cores per walker and different efficiency. The $N_{ind}/Cores$ indicates the ratio of the target number of independent points to the number of cores available for the calculation.

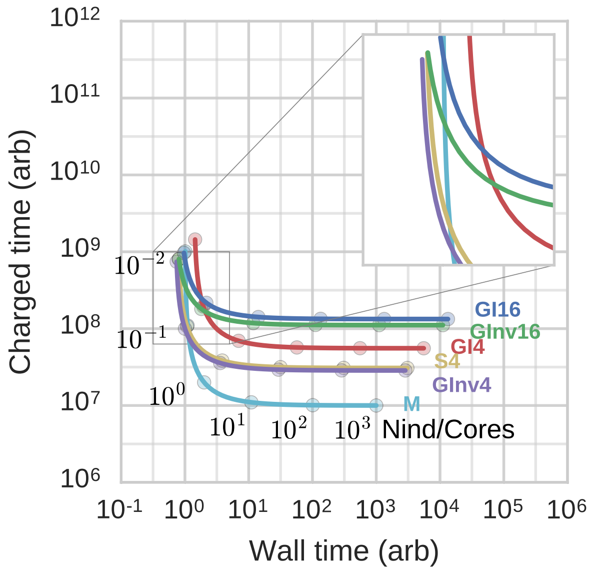

FIG. 5: Charged time vs. Wall time of different algorithms using the measured efficiency. We observe that the standard metropolis algorithm approaches a vertical asymptote as $N_{ind}/Cores$ goes below 1. The calculations become very inefficient beginning at this point.

tal asymtote, the warm-up time becomes negligible compared to the production time. The error bar is large enough that each walker needs to spend significantly more time in production compared to warm-up. Therefore the fixed warm-up time does not affect the total charged time at this point.

As the charged time approaches the vertical asymtote, the warm-up time starts to dominate over the production time. This is when $N_{ind}/Cores < 1$. The error bar becomes too small to allow each walker to have longer production times. Each walker would essentially warm-up and sample a few times before terminating. Since the warm-up time is fixed, the only way to reduce the warm-up time is to improve the sampling process. This is where algorithms that utilize multiple cores per walker at a reduced efficiency can help to reduce the total charged time. We see that algorithms with 4-cores per walker and 16-cores per walker at 90% efficiency reduces the total charged time significantly. Even 16-cores per walker at 10% is able to show some reduction in charged time at high number of cores.

The efficiency estimates we obtained from running calculations for calcium oxychloride allows us to model the performance of the algorithms given various computational resources. FIG. 5 shows the charged time vs. wall time for the different algorithms. We see that at 10^8 cores is where the S_4 and the $GInv_4$ algorithms begin to outperform the standard Metropolis algorithm. At 10^9 we see GI_{16} and $GInv_{16}$ outperform the Metropolis algorithm. This shows that multi-try algorithms can reduce the total charged time of computation when we have high number of total cores.

CONCLUSION

Multiple-try algorithms for QMC can reduce computational time compared to single-try metropolis algorithms, albeit only at the point where $N_{ind}/Cores < 1$. We have shown that it is possible to trade in computational resources for higher sampling efficiency at each step. Production time may be decreased by utilizing more walkers but warm up time is fixed for each walker if the number of core per walker is fixed. Given more cores than the number of independent points, if the sampling process only utilizes one core per walker, the walker would simply warm up and generate one independent point before exiting.

By offering multiple cores per walker, we are able to utilize the extra computational resources more efficiently. Warm up time can be decreased for each walker by utilizing multiple-try algorithms that generate higher number of independent points per step. As we have seen from the measurements, the multiple-try algorithms are not as efficient as the metropolis algorithms. Therefore it is only advantageous to use these algorithms when the target number of independent points is less than the number of cores available.

* pchiu5@illinois.edu
† lkwagner@illinois.edu

[1] D. M. Ceperley and L. Mitas, in *Advances in Chemical Physics*, edited by I. Prigogine and S. A. Rice (John Wiley & Sons, Inc., 1996) pp. 1–38, dOI: 10.1002/9780470141526.ch1.
[2] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal, Reviews of Modern Physics **73**, 33 (2001).
[3] W. K. Hastings, Biometrika **57**, 97 (1970).
[4] A. Mira, *Ordering, slicing and splitting Monte Carlo Markov chains*, Ph.D. thesis, UNIVERSITY OF MINNESOTA (1998).
[5] D. Bressanini, G. Morosi, S. Tarasco, and A. Mira, Journal of Chemical Physics **121**, 3446 (2004).
[6] L. K. Wagner, M. Bajdich, and L. Mitas, Journal of Computational Physics **228**, 3390 (2009).
[7] S. Pandolfi, F. Bartolucci, and N. Friel, Journal of Machine Learning Research **9**, 581 (2010).
[8] J. S. Liu, F. Liang, and Wing Hung Wong, Journal of the American Statistical Association **95**, 121 (2000).
[9] J. Toulouse, R. Assaraf, and C. J. Umrigar, in *Advances in Quantum Chemistry*, Electron Correlation in Molecules – ab initio Beyond Gaussian Quantum Chemistry, Vol. 73, edited by P. E. H. a. T. Ozdogan (Academic Press, 2016) pp. 285–314, dOI: 10.1016/bs.aiq.2015.07.003.
[10] C. J. Umrigar, M. P. Nightingale, and K. J. Runge, The Journal of Chemical Physics **99**, 2865 (1993).