Manager–Worker-Based Model for the Parallelization of Quantum Monte Carlo on Heterogeneous and Homogeneous Networks

MICHAEL T. FELDMANN,[1] JULIAN C. CUMMINGS,[1] DAVID R. KENT IV,[2]
RICHARD P. MULLER,[2,*] WILLIAM A. GODDARD III[2]
[1]*Center for Advanced Computing Research, California Institute of Technology, Pasadena, California 91125*
[2]*Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, California 91125*

Received 10 March 2003; Revised 24 March 2004; Accepted 19 April 2004
DOI 10.1002/jcc.20836
Published online 5 October 2007 in Wiley InterScience (www.interscience.wiley.com).

Abstract: A manager–worker-based parallelization algorithm for Quantum Monte Carlo (QMC-MW) is presented and compared with the pure iterative parallelization algorithm, which is in common use. The new manager–worker algorithm performs automatic load balancing, allowing it to perform near the theoretical maximal speed even on heterogeneous parallel computers. Furthermore, the new algorithm performs as well as the pure iterative algorithm on homogeneous parallel computers. When combined with the dynamic distributable decorrelation algorithm (DDDA) [Feldmann et al., J Comput Chem 28, 2309 (2007)], the new manager–worker algorithm allows QMC calculations to be terminated at a prespecified level of convergence rather than upon a prespecified number of steps (the common practice). This allows a guaranteed level of precision at the least cost. Additionally, we show (by both analytic derivation and experimental verification) that standard QMC implementations are not "perfectly parallel" as is often claimed.

 J Comput Chem 29: 8–16, 2008

Key words: Quantum Monte Carlo; parallelization algorithm; parallel efficiency; initialization; decorrelation

Introduction

There is currently a great deal of interest in making Quantum Monte Carlo (QMC) methods practical for everyday use by chemists, physicists, and material scientists. Since protocols exist for using QMC methods (such as variational QMC, diffusion QMC, and Green's function QMC) to calculate the energy of an atomic or molecular system to within chemical accuracy (<2 kcal/mol), this makes their everyday application very attractive. High accuracy quantum mechanical methods generally scale very poorly with problem size, typically $O(N^6$ to $N!)$, while QMC scales fairly well. Generally QMC scales as $O(N^3)$ with a large prefactor, but recent efforts have reduced the scaling to $O(N)$.[1] Density functional theory (DFT) scales well, $O(N^3)$, and could in principle provide highly accurate solutions, but with the current generation of functionals, DFT typically has an accuracy of only 5 kcal/mol for typical systems. Unfortunately, there is no systematic way to improve the accuracy of DFT.

The primary issue facing the QMC community is that, although QMC scales well with problem size, the prefactor of the method is generally very large, often requiring CPU months to calculate moderately sized systems. The Monte Carlo nature of QMC allows it to be easily parallelized, thus, reducing the prefactor, with respect to the wall clock.

Application of QMC to physically interesting systems almost always requires the use of supercomputers to enable calculations to complete in a reasonable amount of time. Currently, however, supercomputing resources are very expensive and can be difficult

*Present address: Multiscale Computational Materials Methods, Sandia National Laboratories, Albuquerque, New Mexico 87185-1322

Correspondence to: W. A. Goddard III; e-mail: wag@wag.caltech.edu

Contract/grant sponsor: Computational Science Graduate Fellowship Program of the Office of Scientific Computing

Contract/grant sponsor: Office of Defense Program in the Department of Energy; contract/grant number: DE-FG02-97ER25308

Contract/grant sponsor: Fannie and John Hertz Foundation

Contract/grant sponsor: NSF; contract/grant number: CHE-99MRI

Contract/grant sponsors: IOM (SUR Grant), ARO-DURIP (1997), NIH, Chevron-Texaco, 3M, Avery-Dennison, Dow, GM, Trader Joes, Seiko-Epson, Beckman Institute, Asahi Chemical, Nippon Steel

to access. To make QMC more useful for an average practitioner, algorithms must become more efficient, and/or large inexpensive supercomputers must be produced.

A current trend in large scale supercomputing[2] is assembling "cheap supercomputers" with commodity components using a Beowulf-type framework. These clusters have proven to be very powerful for high-performance scientific computing applications.[3] Clusters can be constructed as homogeneous supercomputers if the hardware for each node is equivalent or as heterogeneous supercomputers if various generations of hardware are included.

Another interesting development is the use of loosely coupled, distributed grids of computational resources[4] with components that can even reside in different geographic locations in the world. Such "grids" are upgraded by adding new compute nodes to the existing grid resulting in continuously upgradable supercomputers, which are inevitably heterogeneous. Ultimately, computational grids may provide computational resources on demand just as electrical grids now provide electrical power on demand.

To efficiently utilize the next generation of supercomputer (heterogeneous cluster or grid), a parallelization algorithm must require little communication between processors and must be able to efficiently use processors that are running at different speeds. We propose a manager–worker-parallelization algorithm for QMC (QMC-MW) that is designed for just such systems. This algorithm is compared against the pure iterative parallelization algorithm (QMC-PI), which is most commonly used in QMC implementations.[5–7]

Theory

QMC is often referred to as embarassingly parallalizable and has generated much interest as an application that can be effectively run on extremely large networks. In a parallel QMC calculation, an independent QMC calculation is performed on each processor, and the resulting statistics from all the processors are combined to produce the global result.

QMC calculations can typically be broken into two major computationally expensive phases: initialization and statistics gathering. Points distributed with respect to a complicated probability distribution, in this case the square of the wavefunction amplitude, are required during a QMC calculation of a molecular system. In efficient implementations, this is almost always done using the Metropolis algorithm.[8] The Metropolis algorithm generally uses some points that are perturbed during the course of a calculation that properly sample some underlying distribution. These propagating points are referred to as walkers in this text.

The first points generated by the Metropolis algorithm are not generated with respect to the desired probability distribution so they must be discarded. Additionally, points generated for diffusion QMC and Green's function QMC must be discarded if there are significant excited state contributions which have not yet decayed. This represents the initialization phase. Once the algorithm begins to generate points with respect to the desired distribution, the points are said to be "equilibrated" and can be used to generate valid statistical information for the QMC calculation. This represents the statistics gathering phase and is the phase where useful data are generated.

To obtain statistically independent data, each processor, in a parallel calculation, must perform its own initialization procedure, which is the same length as the initialization procedure on a single processor. When large numbers of processors are used, the fraction of the time devoted to initializing the calculation can be very large and will eventually limit the number of processors that can be effectively used in parallel ("Initialization Catatrophe" Section).

The following two sections analyze theoretically the pure iterative (QMC-PI) and manager–worker (QMC-MW) parallelization algorithms for QMC. The analyses assume that an $O(\log_2(N_{\text{processors}}))$ method, where $N_{\text{processors}}$ is the total number of processors, is used to gather the statistical data from all processors and return it to the root processor which we will refer to as "percolating" the results to the root process.[9] To simplify analysis of the algorithms, the analysis is performed for variational QMC (VMC) with the same number of walkers on each processor, but it is possible to extend the results to other QMC methods.

Pure Iterative Parallelization Algorithm

The pure iterative parallelization algorithm (QMC-PI) is the most commonly implemented parallelization algorithm for QMC (Algorithm A).[5–7] This algorithm has its origins on homogeneous parallel machines and simply allocates an equal fraction of the total work to each processor for both the initialization and gathering phases. Each of the processors execute their required initialization tasks to equilibrate each walker followed by a statistics gathering phase. These processes can then percolate the resultant statistics to the root node once every processor has finished its work.

In this algorithm, the number of QMC steps taken by each processor during the statistics gathering phase, $\text{Steps}_{\text{PI},i}$, is equal to the total number of QMC steps taken for the calculation, $\text{Steps}^{\text{RequiredTotal}}$, divided by the total number of processors, $N_{\text{Processors}}$.

$$\text{Steps}_{\text{PI},i} = \frac{\text{Steps}^{\text{RequiredTotal}}}{N_{\text{Processors}}} \tag{1}$$

The number of QMC steps required to initialize each walker during the initialization, $\text{Steps}^{\text{Initialize}}$, is taken to be a constant. An optimally efficient initialization algorithm would determine how many QMC steps are required to equilibrate each walker, but in current practice, each walker is generally equilibrated for the same number of steps.

The wall clock time required for a QMC calculation using the QMC-PI algorithm, t_{PI}, can be expressed as

$$t_{\text{PI}} = t_{\text{PI},i}^{\text{Initialize}} + t_{\text{PI},i}^{\text{Propagate}} + t_{\text{PI},i}^{\text{Synchronize}} + t_{\text{PI}}^{\text{Communicate}}, \tag{2}$$

where $t_{\text{PI},i}^{\text{Initialize}}$ is the time required to initialize the calculation on processor i, $t_{\text{PI},i}^{\text{Propagate}}$ is the time used in gathering useful statistics on processor i, $t_{\text{PI},i}^{\text{Synchronize}}$ is the amount of time processor i has to wait for other processors to complete their tasks, and $t_{\text{PI}}^{\text{Communicate}}$ is the

wall clock time required to communicate all results to the root node. These components can be expressed in terms of quantities that can be measured for each processor and the network connecting them.

$$t_{\mathrm{PI},i}^{\mathrm{Initialize}} = N_{\mathrm{w}}\left(t_i^{\mathrm{GenerateWalker}} + \mathrm{Steps}^{\mathrm{Initialize}} t_i^{\mathrm{QMC}}\right) \quad (3)$$

$$t_{\mathrm{PI},i}^{\mathrm{Propagate}} = \left(\frac{\mathrm{Steps}^{\mathrm{RequiredTotal}}}{N_{\mathrm{Processors}}}\right) t_i^{\mathrm{QMC}} \quad (4)$$

$$t_{\mathrm{PI}}^{\mathrm{Communicate}} = \log_2(N_{\mathrm{Processors}})(t^{\mathrm{Latency}} + \beta L) \quad (5)$$

Here N_{w} is the number of walkers per processor, $t_i^{\mathrm{GenerateWalker}}$ is the time required to construct a walker on processor i, t_i^{QMC} is the time required for a QMC step on processor i, t^{Latency} is the latency of the network, β is the inverse bandwidth of the network, and L is the amount of data being transmitted between pairs of processors when data is percolated to the root node.

The way this algorithm is constructed, all processors must wait for the slowest processor to complete all of its tasks before the program can terminate. Therefore $t_{\mathrm{PI,slowest}}^{\mathrm{Synchronize}} = 0$, and the wall clock time to complete the QMC-PI calculation is

$$t_{\mathrm{PI}} = t_{\mathrm{PI,slowest}}^{\mathrm{Initialize}} + t_{\mathrm{PI,slowest}}^{\mathrm{Propagate}} + t_{\mathrm{PI}}^{\mathrm{Communicate}}. \quad (6)$$

Furthermore,

$$t_{\mathrm{PI},i}^{\mathrm{Synchronize}} = \left(t_{\mathrm{PI,slowest}}^{\mathrm{Initialize}} + t_{\mathrm{PI,slowest}}^{\mathrm{Propagate}}\right) - \left(t_{\mathrm{PI},i}^{\mathrm{Initialize}} + t_{\mathrm{PI},i}^{\mathrm{Propagate}}\right). \quad (7)$$

Similarly, the total CPU time required for a QMC calculation using the QMC-PI algorithm, T_{PI}, can be expressed as

$$T_{\mathrm{PI}} = T_{\mathrm{PI}}^{\mathrm{Initialize}} + T_{\mathrm{PI}}^{\mathrm{Propagate}} + T_{\mathrm{PI}}^{\mathrm{Synchronize}} + T_{\mathrm{PI}}^{\mathrm{Communicate}}, \quad (8)$$

where $T_{\mathrm{PI}}^{\mathrm{Initialize}}$ is the total time required to initialize the calculation, $T_{\mathrm{PI}}^{\mathrm{Propagate}}$ is the total time used in gathering useful statistics, $T_{\mathrm{PI}}^{\mathrm{Synchronize}}$ is the total time used in synchronizing the processors, and $T_{\mathrm{PI}}^{\mathrm{Communicate}}$ is the total time used to communicate all results to the root node. These components can be expressed in terms of quantities that can be measured for each processor and the network connecting them.

$$T_{\mathrm{PI}}^{\mathrm{Initialize}} = \sum_{i}^{N_{\mathrm{Processors}}} t_{\mathrm{PI},i}^{\mathrm{Initialize}} \quad (9)$$

$$T_{\mathrm{PI}}^{\mathrm{Propagate}} = \sum_{i}^{N_{\mathrm{Processors}}} t_{\mathrm{PI},i}^{\mathrm{Propagate}} \quad (10)$$

$$T_{\mathrm{PI}}^{\mathrm{Synchronize}} = \sum_{i}^{N_{\mathrm{Processors}}} t_{\mathrm{PI},i}^{\mathrm{Synchronize}} \quad (11)$$

$$T_{\mathrm{PI}}^{\mathrm{Communicate}} = (N_{\mathrm{Processors}} - 1)(t^{\mathrm{Latency}} + \beta L) \quad (12)$$

Manager–Worker-Parallelization Algorithm

The manager–worker paradigm (QMC-MW) offers an entirely new method for performing parallel QMC calculations (Algorithm B). This algorithm makes the root node a "manager" and all of the other nodes "workers." Each worker node initializes each of its walkers until each is ready to enter the statistics gathering phase just like with the QMC-PI method. Once in the statistics gathering phase, each processor stores statistics on the calculations it has done. Periodically, the manager will stop performing Monte Carlo steps and will request that each worker percolate a summary of the statistics it has calculated to the manager. Once this is complete, the manager calculates the global statistics of the calculation, and the workers return to performing more Monte Carlo steps. Using the global statistics of the calculation, the manager determines if the calculation is complete based upon user defined convergence specifications. Note that in contrast to QMC-PI, QMC-MW does not *a priori* assign specific ammounts of work to each node, and QMC-MW dynamically terminates based upon the convergence of calculated statistics.

The worker nodes compute Monte Carlo steps until they receive a command from the manager node. The command either tells the worker to (1) percolate its results to the manager node and continue working or (2) percolate its results to the manager node and terminate. The manager periodically collects the statistics that have been calculated. If the statistics are sufficiently converged, the manager commands the workers to send all their data and terminate; otherwise, the manager will do some of its own work and repeat the process again later. Unlike QMC-PI, QMC-MW dynamically determines how much work each processor performs. This allows faster processors to do more work so the calculation is automatically load balanced.

The wall clock time required to perform a QMC-MW calculation can be broken into the same terms as were used for a QMC-PI calculation [eq. (3)].

$$t_{\mathrm{MW}} = t_{\mathrm{MW},i}^{\mathrm{Initialize}} + t_{\mathrm{MW},i}^{\mathrm{Propagate}} + t_{\mathrm{MW},i}^{\mathrm{Synchronize}} + t_{\mathrm{MW},i}^{\mathrm{Communicate}} \quad (13)$$

Because MW dynamically determines how many steps are performed by each processor, each of the constituent terms has a more complicated form than in QMC-PI. Allowing $\hat{\tau}$ to be the minimum wall clock needed to achieve convergence on a given network and τ to be the approximate wall clock time minus communication time during the run, one can easily derive the following expressions. Once τ plus communication time exceeds $\hat{\tau}$, the QMC-MW algorithm will terminate.

$$t_{\mathrm{MW},i}^{\mathrm{Initialize}} = N_{\mathrm{w}} t_i^{\mathrm{GenerateWalker}} + \mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Initialize}}(\hat{\tau}) t_i^{\mathrm{QMC}} \quad (14)$$

$$t_{\mathrm{MW},i}^{\mathrm{Propagate}} = \mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Propagate}}(\hat{\tau}) t_i^{\mathrm{QMC}} \quad (15)$$

$$t_{\mathrm{MW},i}^{\mathrm{Communicate}} = \left\lceil \frac{\mathrm{Steps}_{\mathrm{MW},0}^{\mathrm{Total}}(\hat{\tau})}{N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Reduce}}} \right\rceil \log_2(N_{\mathrm{Processors}})(t^{\mathrm{Latency}} + \beta L) + \left\lceil \frac{\mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Total}}(\hat{\tau})}{N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Poll}}} \right\rceil t_i^{\mathrm{Poll}} \quad (16)$$

$$t_{\mathrm{MW},i}^{\mathrm{Synchronize}} \leq N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Poll}} t_{\mathrm{slowest}}^{\mathrm{Poll}} \left\lceil \frac{\mathrm{Steps}_{\mathrm{MW},0}^{\mathrm{Total}}(\hat{\tau})}{N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Reduce}}} \right\rceil, \quad (17)$$

where

$$\tau = t_{\mathrm{MW}} - \left\lceil \frac{\mathrm{Steps}_{\mathrm{MW},0}^{\mathrm{Total}}(\tau)}{N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Reduce}}} \right\rceil \log_2(N_{\mathrm{Processors}})(t^{\mathrm{Latency}} + \beta L) \quad (18)$$

$$\approx t_{\mathrm{MW}} - t_{\mathrm{MW},i}^{\mathrm{Synchronize}} - t_{\mathrm{MW},i}^{\mathrm{Communicate}}$$

$$= t_{\mathrm{MW},i}^{\mathrm{Initialize}} + t_{\mathrm{MW},i}^{\mathrm{Propagate}}$$

$$\mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Total}}(\tau) = \left\lceil \frac{\tau}{N_{\mathrm{w}} t_i^{\mathrm{QMC}}} \right\rceil, \quad (19)$$

$$\mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Initialize}}(\tau) = \min\left(\mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Total}}(\tau), N_{\mathrm{w}}\mathrm{Steps}_{\mathrm{Initialize}}\right), \quad (20)$$

$$\mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Propagate}}(\tau) = \mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Total}}(\tau) - \mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Initialize}}(\tau), \quad (21)$$

and

$$\hat{\tau} = \min \tau \ni \begin{cases} \sum_i^{N_{\mathrm{Processors}}} \mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Propagate}}(\tau) \geq \mathrm{Steps}^{\mathrm{RequiredTotal}} \\ \tau/\left(\mathrm{Steps}^{\mathrm{Reduce}} t_0^{\mathrm{QMC}}\right) \in \mathbf{Z}^+ \end{cases}. \quad (22)$$

$\mathrm{Steps}^{\mathrm{RequiredTotal}}$ is the minimum number of steps that are required to obtain the desired level of convergence, $\mathrm{Steps}^{\mathrm{Poll}}$ is the number of QMC steps that take place on a worker processor between checking for a message from the manager, and $\mathrm{Steps}^{\mathrm{Reduce}}$ is the number of QMC steps that take place on the manager processor between sending commands to the workers. Unlike t_{PI}, t_{MW} cannot be simply expressed in terms of individual processor speeds.

The total time required for the MW algorithm, T_{MW}, can be expressed as

$$T_{\mathrm{MW}} = T_{\mathrm{MW}}^{\mathrm{Initialize}} + T_{\mathrm{MW}}^{\mathrm{Propagate}} + T_{\mathrm{MW}}^{\mathrm{Synchronize}} + T_{\mathrm{MW}}^{\mathrm{Communicate}}, \quad (23)$$

which contains the same components as eq. (9).

$$T_{\mathrm{MW}}^{\mathrm{Initialize}} = \sum_i^{N_{\mathrm{Processors}}} t_{\mathrm{MW},i}^{\mathrm{Initialize}} \quad (24)$$

$$T_{\mathrm{MW}}^{\mathrm{Propagate}} = \sum_i^{N_{\mathrm{Processors}}} t_{\mathrm{MW},i}^{\mathrm{Propagate}} \quad (25)$$

$$T_{\mathrm{MW}}^{\mathrm{Synchronize}} = \sum_i^{N_{\mathrm{Processors}}} t_{\mathrm{MW},i}^{\mathrm{Synchronize}} \quad (26)$$

$$T_{\mathrm{MW}}^{\mathrm{Communicate}} = \left\lceil \frac{\mathrm{Steps}_{\mathrm{MW},0}^{\mathrm{Total}}}{N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Reduce}}} \right\rceil (N_{\mathrm{Processors}} - 1)(t^{\mathrm{Latency}} + \beta L) + \sum_i^{N_{\mathrm{Processors}}} \left\lceil \frac{\mathrm{Steps}_{\mathrm{MW},i}^{\mathrm{Total}}(\hat{\tau})}{N_{\mathrm{w}}\mathrm{Steps}^{\mathrm{Poll}}} \right\rceil t_i^{\mathrm{Poll}} \quad (27)$$

Initialization Catastrophe

QMC algorithms are described as being "embarrassingly parallel" and linearly scaling with respect to the number of processors used.[10] These statements are true for a large fraction of Monte Carlo calculations but are *not* true for QMC calculations which employ the Metropolis algorithm.[8]

To obtain independent statistical data from each processor, at least one independent Markov chain must be initialized on each processor ("Theory" section). This gives an initialization cost, $T^{\mathrm{Initialize}}$, which scales as $O(N_{\mathrm{Processors}})$. The time devoted to generating useful statistical data during the calculation, $T^{\mathrm{Propagate}}$, scales as $O(1)$ because a given number of independent Monte Carlo samples are required to obtain a desired statistical accuracy no matter how many processors are used. From this, the efficiency, or fraction of the total calculation time devoted to useful work, ϵ is

$$\epsilon = \frac{T^{\mathrm{Propagate}}}{T^{\mathrm{Initialize}} + T^{\mathrm{Propagate}} + T^{\mathrm{Synchronize}} + T^{\mathrm{Communicate}}} \quad (28)$$

$$\approx \frac{O(1)}{O(N_{\mathrm{Processors}}) + O(1)}. \quad (29)$$

This result clearly demonstrates that QMC calculations using the Metropolis algorithm are *not* linearly scaling for large numbers of processors as is often claimed. This results from the initialization of the Metropolis algorithm and not the parallelization algorithm used.

For QMC calculations to efficiently use $>10^4$ processors, new algorithms to efficiently generate equilibrated, statistically independent walkers are required. The effort to generate such walkers for the global calculation scales linearly with the number of processors because $N_{\mathrm{Processors}}N_{\mathrm{w}}$ walkers are required. Thus, the initialization catastrophe cannot be eleminated but can be minimized.

Experiment

Computational experiments comparing QMC-PI and QMC-MW parallelization algorithms were performed using *QMcBeaver*,[11–13] a finite all-electron QMC software package we developed. Variational QMC was chosen as the particular QMC flavor to allow direct comparison with the theoretical results in the previous section.

QMcBeaver percolates statistical results from all nodes to the root node using the dynamic distributable decorrelation algorithm (DDDA)[12,13] and the *MPI_Reduce* command from MPI.[14] This combination provides an $O(\log_2(N_{\mathrm{Processors}}))$ method for gathering the statistical data from all processors, decorrelating the statistical data, and returning it to the root node.

The time spent initializing, propagating, synchronizing, and communicating during a calculation was obtained from timers inserted into the relevant sections of *QMcBeaver*. During a parallel calculation, each node has its own set of timers which provide information on how that particular processor is performing. At the completion of a calculation, the results from all processors are combined to yield the total CPU time devoted to each class of task.

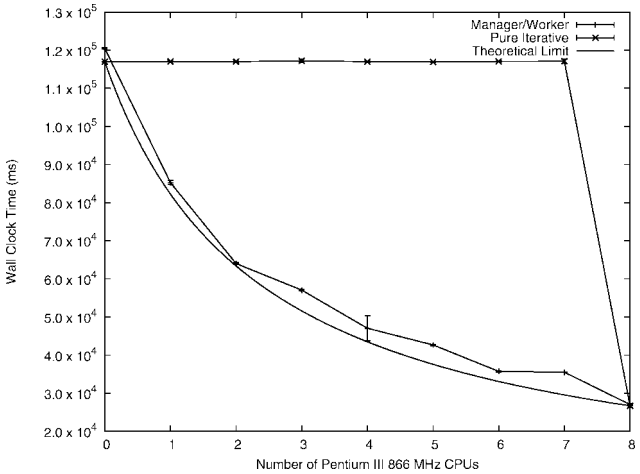

Figure 1. Time required to complete an eight processor variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms. The eight processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking. The theoretical optimal performance for a given configuration of processors is provided by the curve.

Experiment: Varying Levels of Heterogeneity

For this experiment, a combination of Intel Pentium Pro 200 MHz and Intel Pentium III 866 MHz computers connected with a 100 Mb/sec network was used. The total number of processors was kept constant at eight, but the number of each type of processor was varied over the whole range. This setup provided a series of eight processor parallel computers with a spectrum of heterogeneous configurations. For our calculations with the current version of *QMcBeaver*, the Pentium III is roughly 4.4 times faster than the Pentium Pro at performing QMcBeaver on these test systems.

Variantional QMC computational experiments were performed on a Ne atom using a Hartree–Fock/TZV[15] trial wavefunction calculated using GAMESS.[16,17] For the parallelization algorithms, the following values were used: $\text{Steps}^{\text{RequiredTotal}} = 2.5 \times 10^6$, $\text{Steps}^{\text{Initialize}} = 1 \times 10^3$, $\text{Steps}^{\text{Poll}} = 1$, $\text{Steps}^{\text{Reduce}} = 1 \times 10^3$, and $N_w = 2$.

The time required to complete the QMC calculation for the QMC-PI and QMC-MW parallelization algorithms is shown in Figure 1. Each data point was calculated five times and averaged to provide statistically relevant data.

The time required for the QMC-PI algorithm to complete is determined by the slowest processor. When between one and eight Pentium Pro processors are used, the calculation takes the same time as when eight Pentium Pro processors are used; yet, when eight Pentium III processors are used (homogeneous network), the calculation completes much faster. This matches the behavior predicted by eq. (6). This figure also shows that MW performs near the theoretical speed limit for each of the heterogeneous configurations. This is a result of the dynamic load balancing inherent in QMC-MW.

The total number of QMC steps performed during a calculation is shown in Figure 2. The QMC-PI method executes the same number of steps regardless of the particular network because the number of steps performed by each processor is determined *a priori*. On the other hand, QMC-MW executes a different number of steps for each network configuration. This results from the dynamic determination of the number of steps performed by each processor. The total number of steps is always greater than or equal to the number of steps needed to obtain a desired precision, $\text{Steps}^{\text{RequiredTotal}}$.

Figures 3 and 4 break the total calculation time down into its constituent components [eqs. (8) and (23)]. QMC-MW spends essentially all of its time initializing walkers or generating useful QMC data. Synchronization and communication costs are minimal. On the other hand, QMC-PI devotes a huge portion of the total calculation time to synchronizing processors on heterogeneous networks.

In Figure 3 one notices that the maximum inefficiency for QMC-PI comes with seven fast machines and one slow machine and not at the point of maximum heterogenity. Equations 6 and 7 show that the total time of the calculation is determined by the slowest processor. Thus the lowest total efficiency for QMC-PI results, given a constant slowest processor, when the computing power is at a maximum. In this set of experiments, when the slowest processor is a Pentium Pro, the maximum computing power is achieved with one Pentium Pro and seven Pentium III processors; however, as explained earlier, this configuration has the lowest efficiency for QMC-PI.

One should note that the value of $\text{Steps}^{\text{RequiredTotal}}$ required to obtain a desired precision in the calculated quantities is not known before a calculation begins. QMC-PI requires this value be estimated *a priori*. If the guess is too large, the calculation is converged beyond what is required, and if too small, the calculation must be restarted from a checkpoint. Both situations are inefficient. Because QMC-MW does not determine $\text{Steps}^{\text{RequiredTotal}}$ *a priori*, an optimal value can be determined "on-the-fly" by examining the convergence of the

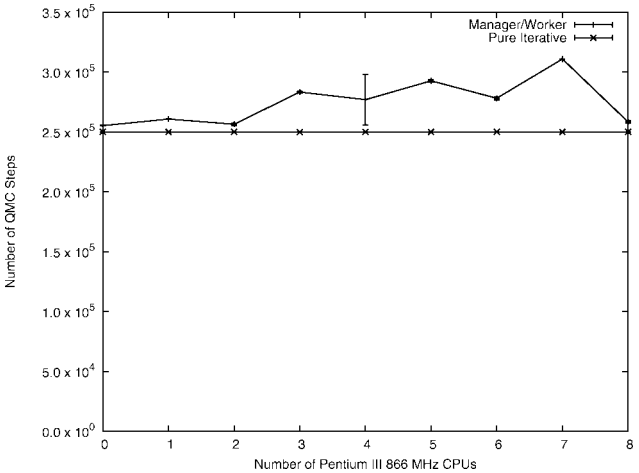

Figure 2. Number of variational QMC steps completed during an eight processor calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) parallelization algorithms. The pure iterative algorithm always calculates the same number of steps, but the manager–worker algorithm dynamically determines how many steps to take. The eight processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking.

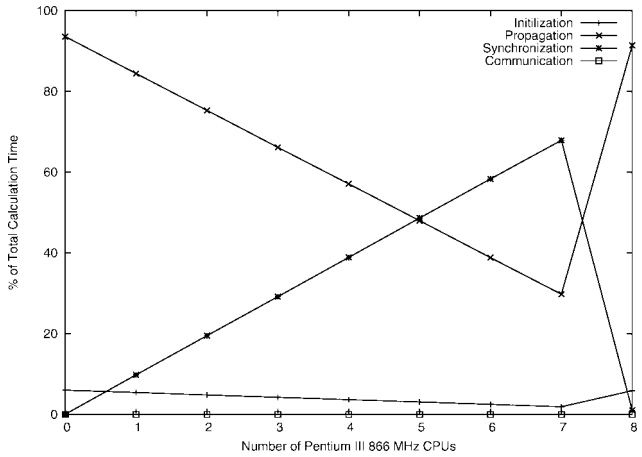

Figure 3. Percentage of total calculation time devoted to each component in the pure iterative parallelization algorithm (QMC-PI) during an eight processor variational QMC calculation of Ne. The eight processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking.

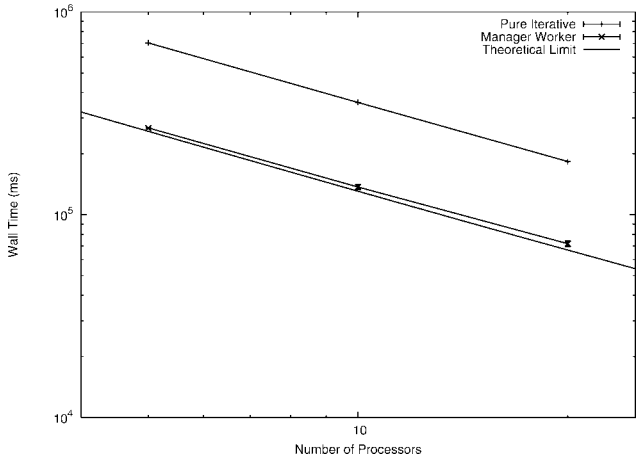

Figure 5. Wall time required to complete a variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms on a heterogeneous Linux cluster. The theoretical optimal performance for a given configuration of processors is provided by the line. The specific processor configuration is discussed in Experiment section.

calculation. This provides the outstanding performance of QMC-MW on any architecture.

Experiment: Heterogeneous Network Size

Variantional QMC computational experiments were performed on a Ne atom using a Hartree-Fock/TZV[15] trial wavefunction calculated using GAMESS.[16,17] The network of machines used was a heterogeneous cluster of Linux boxes. The five processor data point was generated using an Intel Pentium Pro 200 MHz, Intel Pentium II 450 MHz, Intel Pentium III Xeon 550 MHz, Intel Pentium III 600 MHz, and Intel Pentium III 866 MHz. The 10 and 20 processor data points represent two and four times as many processors, respectively, with the same distribution of processor types as the five processor data point. All computers are connected by 100 Mb/sec networking. For the parallelization algorithms, the following values were used: $\text{Steps}^{\text{RequiredTotal}} = 2.5 \times 10^6$, $\text{Steps}^{\text{Initialize}} = 1 \times 10^3$, $\text{Steps}^{\text{Poll}} = 1$, $\text{Steps}^{\text{Reduce}} = 1 \times 10^3$, and $N_{\text{w}} = 2$.

The time required to complete the QMC calculation for the QMC-PI and QMC-MW parallelization algorithms is shown in Figure 5. Each data point was calculated five times and averaged to provide statistically relevant data.

The results illustrate that QMC-MW performs near the theoretical performance limit as the size of a heterogeneous calculation increases. Because all three data points were calculated with an Intel Pentium Pro 200 MHz as the slowest processor, the QMC-PI calculations perform like 5, 10, and 20 processor Pentium Pro 200 MHz calculations. The scaling is essentiallly linear, but there is a huge inefficiency illustrated by the separation between the theoretical performance limit and the QMC-PI results. Ideally, Figure 5 should include data for very large networks of machines in the particular ratios used here. Unfortunately, the cluster used here was the largest available. For large scaling runs please refer to later sections in this text (following two sections).

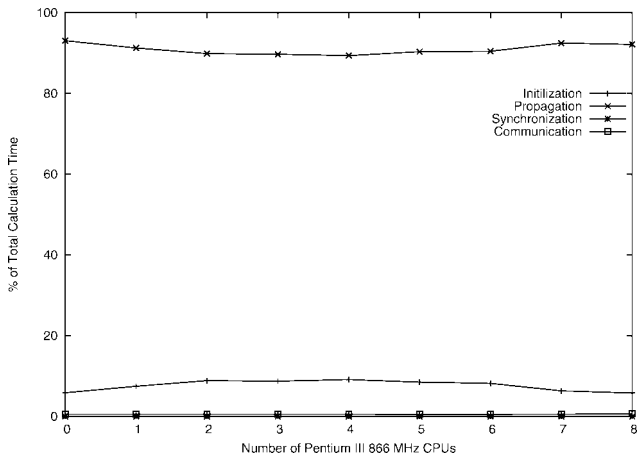

Figure 4. Percentage of total calculation time devoted to each component in the manager–worker-parallelization algorithm (QMC-MW) during an eight processor variational QMC calculation of Ne. The eight processors are a mixture of Pentium Pro 200 MHz and Pentium III 866 MHz Intel processors connected by 100 Mb/s networking.

Experiment: Large Heterogeneous Network

Variantional QMC computational experiments were performed on NH_2CH_2OH using a B3LYP(DFT)/cc-pVTZ[18] trial wavefunction calculated using Jaguar 4.0.[19] The calculations were run on the parallel distributed systems facility (PDSF) at the national energy research scientific computing center (NERSC). This machine is a heterogeneous cluster of Linux boxes with Intel processors ranging from Pentium II 400 MHz to Pentium III 1 GHz.

Two calculations were performed on the cluster. The first used 128 processors with an average processor clock speed of 812 MHz, and the second used the whole cluster, 355 processors, with an average processor clock speed of 729 MHz. Both calculations were

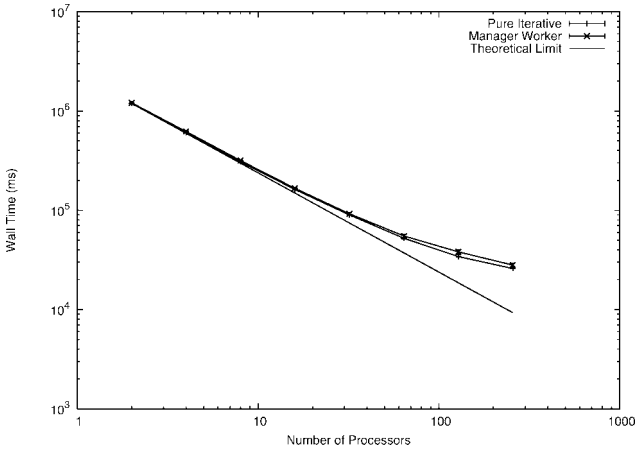

Figure 6. Wall time required to complete a variational QMC calculation of Ne using the manager–worker (QMC-MW) and pure iterative (QMC-PI) algorithms on the ASCI Blue Pacific homogeneous supercomputer. The theoretical optimal performance for a given configuration of processors is provided by the line.

done using only the QMC-MW algorithm. For the parallelization algorithm, the following values were used: $\text{Steps}^{\text{RequiredTotal}} = 1\times10^7, \text{Steps}^{\text{Initialize}} = 2\times10^3, \text{Steps}^{\text{Poll}} = 1, \text{Steps}^{\text{Reduce}} = 1\times10^4$, and $N_w = 1$.

The 128 and 355 processor calculations completed in 6.427×10^3 s and 2.645×10^3 s, respectively. These results can be crudely compared by assuming that all processors used in the calculation perform the same amount of work per clock cycle. Using this assumption, the calculation is 98% efficient in scaling up from 128 to 355 processors. This demonstrates the ability of QMC-MW to efficiently deal with very large heterogeneous computers.

Experiment: Homogeneous Network

The QMC-PI algorithm was originally designed to work on homogeneous supercomputers with fast communication while the QMC-MW algorithm was designed to work on heterogeneous supercomputers with slow communication. To test the QMC-MW algorithm on the QMC-PI algorithm's native architecture, a QMC scaling calculation (Fig. 6) was performed on the ASCI-Blue Pacific supercomputer at Lawrence Livermore National Laboratory. This machine is a homogeneous supercomputer composed of 332 MHz PowerPC 604e processors connected by HIPPI networking.

Variantional QMC computational experiments were performed on a Ne atom using a Hartree-Fock/TZV[15] trial wavefunction calculated using GAMESS.[16,17] For the parallelization algorithms, the following values were used: $\text{Steps}^{\text{RequiredTotal}} = 1 \times 10^6$, $\text{Steps}^{\text{Initialize}} = 2 \times 10^3$, $\text{Steps}^{\text{Poll}} = 1$, $\text{Steps}^{\text{Reduce}} = 1 \times 10^3$, and $N_w = 2$.

Figure 6 shows that the QMC-MW and QMC-PI algorithms perform nearly identically on Blue Pacific. The QMC-MW calculation is consistently slightly slower than the QMC-PI algorithm because the QMC-MW calculation performed more QMC steps. This results because the QMC-PI calculation performs a predetermined number of steps while the QMC-MW calculation performs *at least* this same predetermined number of steps for this experiment. Again, this assumes the user knows *a priori* exactly how many steps to complete for QMC-PI, in order to obtain the desired convergence, while QMC-MW requires no *a priori* knowledge of $\text{Steps}^{\text{RequiredTotal}}$. This experiment is an absolute best case situation for the QMC-PI algorithm. This discrepancy can be reduced by decreasing $\text{Steps}^{\text{Reduce}}$.

Figure 7 plots the ratio of the total computational resources used by each algorithm. This shows that both algorithms perform within 2% of each other; therefore, they can be considered to take roughly the same time and expense on homogeneous machines. Since all of these ratios are very near 1.0 (which is predicted from the earlier sections for homogeneous machines with a given number of statistic gathering steps) the minor fluxuations about 1.0 are statistical and of no concern.

Both algorithms do not perform near the linear scaling limit for large numbers of processors. This is a result of the initialization catastrophe discussed in earlier sections.

Experiment: Initialization Catastrophe

To demonstrate the "initialization catastrophe" described in "Intialization Catastrophe" section, a scaling experiment was performed on the ASCI-Nirvana supercomputer at Los Alamos National Laboratory (Fig. 8). This machine is a homogeneous supercomputer composed of 2048 MIPS 10,000 processors running at 250 MHz connected by HIPPI networking. Variational QMC calculations of RDX, cyclic-$[CH_2NNO_2]_3$, using the QMC-MW algorithm with $\text{Steps}^{\text{RequiredTotal}} = 1 \times 10^5$, $\text{Steps}^{\text{Initialize}} = 1 \times 10^3$, $\text{Steps}^{\text{Poll}} = 1$, $\text{Steps}^{\text{Reduce}} = 1 \times 10^2$, and $N_w = 1$ were performed. Jaguar 4.0[19] was used to generate a HF/6-31G** trial wavefunction.

The efficiency of the scaling experiments were calculated using eq. (28), and the results were fit to

$$\epsilon = \frac{a}{1 + bN_{\text{Processors}}} \tag{30}$$

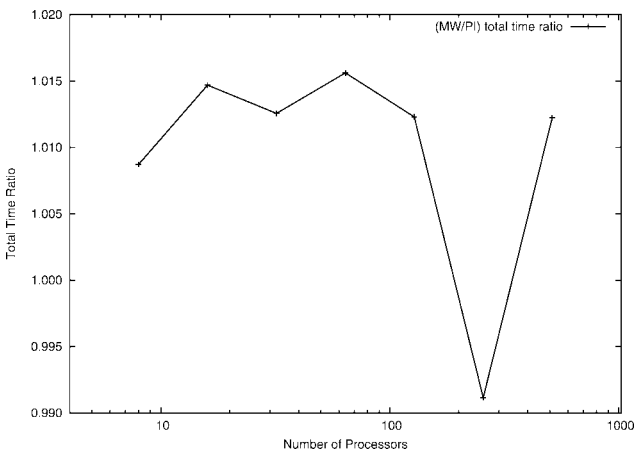

Figure 7. Ratio of wall time for QMC-MW/QMC-PI on ASCI Blue Pacific.

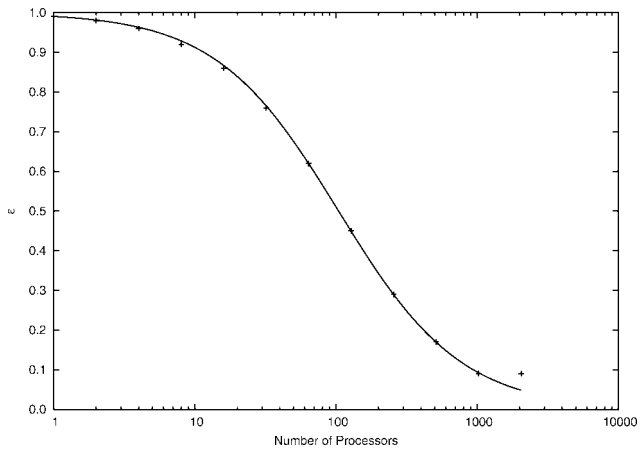

Figure 8. Efficiency of a variational QMC calculation of RDX as a function of the number of processors used. The calculations were performed using the manager–worker-parallelization algorithm (QMC-MW) on the ASCI-Blue Mountain supercomputer, which has 250 MHz MIPS 10,000 processors connected by HIPPI networking. A similar result is produced by the Pure Iterative parallelization algorithm. The data is fit to $\epsilon(N_{\text{Processors}}) = a/(1 + bN_{\text{Processors}})$ with $a = 0.9954$ and $b = 0.0095$.

with $a = 0.9954$ and $b = 0.0095$ (This is equivalent to fitting to $\epsilon = d/(e+fN_{\text{Processors}})$ when a common factor of e is removed from the numerator and denominator.). The efficiency at 2048 processors is better than the value predicted from the fit equation. This is an artifact of the QMC-MW algorithm which resulted from this calculation taking significantly more steps than $\text{Steps}^{\text{RequiredTotal}}$. Decreasing the value of $\text{Steps}^{\text{Reduce}}$ would reduce this problem.

The excellent fit of the data to eq. (30) clearly shows that QMC calculations using the Metropolis algorithm are *not* linearly scaling for large numbers of processors. This result holds true for both QMC-MW and QMC-PI because it results from the initialization of the Metropolis algorithm and not the parallelization of the statistics gathering propagation phase. Furthermore, longer statistics gathering calculations have better efficiencies and thus better scaling than short statistics gathering calculations. This can be seen by examining eq. (28).

Conclusion

The new QMC manager–worker-parallelization algorithm clearly outperforms the commonly used Pure Iterative parallelization algorithm on heterogeneous parallel computers and performs near the theoretical speed limit. Furthermore, both algorithms perform essentially equally well on a homogeneous supercomputer with high speed networking.

When combined with DDDA, QMC-MW is able to determine, "on-the-fly," how well a calculation is converging, allowing convergence-based termination. This is opposed to the standard practice of having QMC calculations run for a predefined number of steps. If the predefined number of steps is too long, computer time is wasted, and if too short, the job will not have the required convergence and must be resubmitted to the queue lengthening the total time for the calculation to complete. Additionally, specifying a calculation precision (2 kcal/mol for example) is more natural for the application user than specifying a number of QMC steps.

QMC-MW allows *very* low cost QMC specific parallel computers to be built. These machines can use commodity processors, commodity networking, and no hard disks. Because the algorithm efficiently handles loosely coupled heterogeneous machines, such a computer is continuously upgradeable and can have new nodes added as resources become available. This greatly reduces the cost of the resources the average practitioner needs access to, bringing QMC closer to becoming a mainstream method.

It is possible to use QMC-PI on a heterogeneous computer with good efficiency if the QMC performance on each processor is known. Determining and effectively using this information can be a great deal of work. If the user has little or inaccurate information about the computer, this approach will fail. QMC-MW overcomes these shortfalls with no work or input on the users part. Also, when new nodes are added to the computer, QMC-MW can immediately take advantage of them whereas the modified QMC-PI must have benchmark information recorded before they can be efficiently used. The benefits and displayed ease of implementation of QMC-MW clearly outweigh those of QMC-PI supporting its adoption as the method of choice for making QMC parallel.

For calculations with $>10^4$ processors, a modification to the presented QMC-MW algorithm could yield a large performance increase. This modification involves two threads of execution per processor. A lightweight "listener" thread would manage all of the communication between the manager and worker nodes while a heavyweight thread would perform the actual QMC calculation.

The prediction and verification of the initialization catastrophe clearly highlights the need for efficient initialization schemes if QMC is to be scaled to tens of thousands or more processors. Current initialization schemes initialize each walker for the same number of steps because very little research has been done to determine when a walker is equilibrated and correctly samples the desired probability distribution. Thus $\text{Steps}^{\text{Initialize}}$ is typically larger than necesary to equilibrate many of the walkers. The development of algorithms that can identify when a walker is equilibrated will reduce the total number of steps required to initialize the calculation, thus minimizing the initialization catastrophe. In addition, algorithms to more effectively pick initial guess walkers will reduce the required number of equilibration steps further. Producing such algorithms is challenging and must be a focus of future work.

Acknowledgments

The computational resources at LANL and LLNL were under the DOE ASCI ASAP project at Caltech. Special thanks go to Shane Canon and Iwona Sakrejda for providing dedicated computer time on PDSF at NERSC.

Appendix A: Pure Iterative Algorithm (QMC-PI)

for Processor_i; $i = 0$ to $N_{\text{Processors}} - 1$
    $\text{Steps}_{\text{PI},i} = \text{Steps}^{\text{RequiredTotal}}/N_{\text{Processors}}$
    Generate N_{w} walkers

```
for Steps^Initilize steps
    Equilibrate walkers
for Steps_PI,i steps
    Generate QMC statistics
Percolate statistics to Processor_0
```

Appendix B: Manager–Worker Algorithm (QMC-MW)

```
for Processor_i; i = 0 to N_Processors − 1
    done = false
    counter = 0
    Generate N_w walkers

    while not done:
        if counter < Steps^Initialize:
            Equilibrate all local walkers 1 step
        else:
            Propagate all local walkers 1 step and collect
            QMC statistics

        if i = 0:
            if statistics are converged:
                done = true
                Tell workers to percolate statistics to
                Processor_0 and
                set done = true
            else if counter mod Steps^Reduce = 0:
                Tell workers to percolate statistics to
                Processor_0

        else:
            if counter mod Steps^Poll = 0:
                Check for commands from the manager
                and
                execute the commands.

    counter = counter + 1
```

References

1. Williamson, A.; Hood, R.; Grossman, J. Phys Rev Lett 2001, 87, 246406.
2. Yaser, O. Parallel Comput, 2001, 27, 1.
3. Deng, Y.; Korobka, A. Parallel Comput 2001, 27, 91.
4. Foster, I.; Kesselman, C.; Tuecke, S. Int J High Perform Comput Appl 2001, 15, 200.
5. Muller, R. P.; Feldmann, M. T.; Barnett, R. N.; Hammond, B. L.; Reynold, P. J.; Terray, L.; Lester, W. A. Jr., California Institute of Technology, Pasadena, CA 91125, Material Simulation Center parallel QMAGIC, version 1.1.0p; 2000.
6. Needs, R.; Rajagopal, G.; Towler, M. D.; Kent, P. R. C.; Williamson, A. CASINO, the Cambridge Quantum Monte Carlo code, version 1.1.0; University of Cambridge, Cambridge, UK, 2000.
7. Smith, L.; Kent, P. Concurrency: Pract Exp 2000, 12, 1121.
8. Metropolis, N.; Rosenbluth, A. W.; Rosenbluth, M. N.; Teller, A. H.; Teller, E. J Chem Phys 1953, 21, 1087.
9. Feldmann, M. T.; Kent, D. R., IV; Anderson, A. G.; Muller, R. P.; Goddard, W. A., III. J Comput Chem 2007, 28, 2309.
10. Kent, P. R. C. Ph.D. Thesis, Robinson College at Cambridge; 1999.
11. Feldmann, M. T.; Kent, D. R., IV. QM^{c}Beaver v20020109 ©(2001–2002).
12. Feldmann, M. T. Ph.D. Thesis, California Institute of Technology, Pasadena, California 2002.
13. Kent, D. R., IV, Ph.D. Thesis, California Institute of Technology, Pasadena, California; 2003.
14. Snir, M.; Otto, S.; Huss-Lederman, S.; Walker, D.; Dongarra, J. MPI – The Complete Reference Vol. 1: The MPI Core; The MIT Press: Cambridge, MA, 1998; 2nd ed., ISBN 0-262-69215-5.
15. Dunning, T. H. Jr. J Chem Phys 1971, 55, 716.
16. Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; et al. J Comput Chem 1993, 14, 1347.
17. Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M.; Montgomery, J. A. Jr. J Comput Chem 1993, 14, 1347. http://dx.doi.org/10.1002/jcc.540141112.
18. Dunning, T. J Chem Phys 1989, 90, 1007.
19. Ringnalda, M. N.; Langlois, J.-M.; Murphy, R. B.; Greeley, B. H.; Cortis, C.; Russo, T. V.; Marten, B.; Donnelly, R. E., Jr.; Pollard, W. T.; Cao, Y. et al., Jaguar v4.0, 2001.