QMCPACK Tutorial

Jeongnim Kim
& QMCPACK developers
http://qmcpack.cmscc.org/

Acknowledgements

QMCPACK developers*

- Kenneth P. Esler (Stoneridge)
- Jeremy McMinis (UI)
- Miguel Morales (LLNL)
- Bryan Clark (Princeton)
- Luke Shulenburger (Sandia)
- Simone Chiesa (W&M)
- Kris Delaney (UCSB)
- Jaron Krogel (UI)

and more

*http://qmcpack.cmscc.org/

QMC Endstation

- David M Ceperley (UI)
- S. Zhang & H. Krakauer (W&M)
- P. Kent (ORNL)
- L. Mitas (NCSU)
- Umrigar & Hennig (Corrnell)
- A. Srinivasan (FSU)

Special thanks to

- T. C. Schulthess (ORNL, CSCS)
- Richard M. Martin (UI)
- John W. Wilkins (OSU)

QMCPACK: QMC for HPC

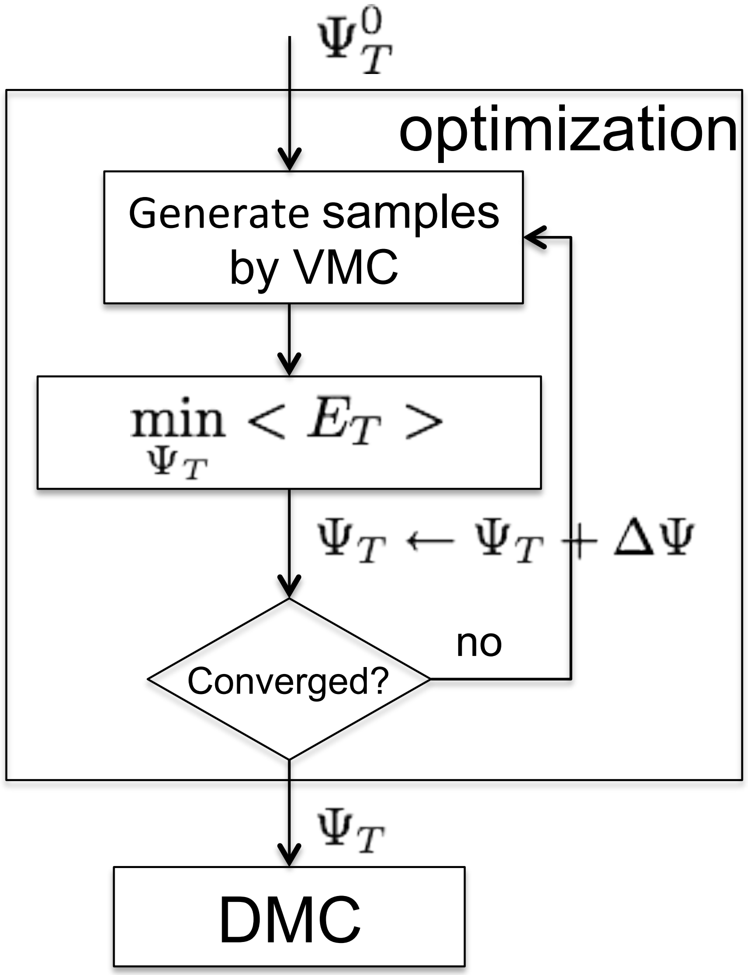
- Implements essential QMC algorithms and best practices developed over 20yrs+
- Designed for large-scale QMC simulations of molecules, solids and nanostructures on massively parallel machine
  - (OpenMP,CUDA)/MPI Hybrid parallelization
  - Object-oriented and generic programming in C++
- Apply software engineering
  - Reusable and extensible solution for new development
  - Standard open-source libraries and utilities for development, compilation and execution
  - Portable and scalable I/O with XML/HDF5

http://qmcpack.cmscc.org/

Key features

QMC Methods	Many-body Wavefunction	Many-body Hamiltonian
• VMC, DMC, RMC • Correlated sampling in VMC,RMC • Optimization based on linear method: energy and variance minimization	• 1,2, 3-body & k-space Jastrow • Single Slater-Jastrow • Multi Slater-Jastrow • Backflow • AGP in LCAO	• Kinetic operator • Coulomb potential • Pseudopotentials • Multiple forms of He • Model Hamiltonian on 3D grid
Programming and platforms	**Single-particle orbitals**	**Finite-size corrections**
• C++ and CUDA • Object-oriented, generic programming and design patterns • (OpenMP,CUDA)/MPI parallelization • Ported on clusters of SMP: x86, IBM Power and Blue Gene, etc	• Plane Wave • Linear combination of atomic orbitals (LCAO): Gaussian, Slater, numerical and mixed basis • B-spline (einspline) • LAPW with B-spline	• Twist averaging • Chiesa correction • Modified potential correction

QMC Workflow



- Initial guess Ψ_T^0
  - Compact, easy to evaluate, but close to true Ψ

$$\Psi_T(\mathbf{R}) = J(\boxed{\{\alpha\}}) \sum \boxed{C_i} D_i^{\uparrow}(\phi) D_i^{\downarrow}(\phi)$$

  - Single-particle orbitals $\boxed{\{\phi\}}$
    e.g., KS or HF solution
- Find $\{\alpha\}$ & $\{C\}$ to optimize an object function: energy and variation minimization
- Projecting out the ground-state by applying a propagator $e^{-\tau\hat{H}}$

Core Computations

For each walker,

All about Ψ_T

```
let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
for particle $i = 1 \cdots N$ do
    set $\mathbf{r}_i^{'} = \mathbf{r}_i + \delta$
    let $\mathbf{R}^{'} = \{\mathbf{r}_1 \ldots \mathbf{r}_i^{'} \ldots \mathbf{r}_N\}$
    ratio $\rho = \Psi_T(\mathbf{R}^{'})/\Psi_T(\mathbf{R})$
    if $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted then
        update state of a walker
    end if
end for{particle}
Compute $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
```

$$\delta = r + \tau \nabla_i \ln \Psi_T$$

Quantum force

$$\frac{\Psi_T(\mathbf{r}_1 \cdots \mathbf{r}_i^{'} \cdots \mathbf{r}_N)}{\Psi_T(\mathbf{r}_1 \cdots \mathbf{r}_i \cdots \mathbf{r}_N)}$$

$$\Psi_T \leftarrow \Psi_T(\mathbf{r}_1 \cdots \mathbf{r}_i^{'} \cdots \mathbf{r}_N)$$

$$f(\{\mathbf{R}\}, \nabla \ln \Psi_T, \nabla^2 \ln \Psi_T)$$

Use $\Psi_T = \Pi_i \Psi_i \longrightarrow \ln \Psi_T = \sum_i \ln \Psi_i$

Slater-Jastrow for Electrons

$$\Psi_T(\mathbf{R}) = e^{J_1+J_2+\cdots} \sum C_i D_i^{\uparrow}(\phi) D_i^{\downarrow}(\phi) \qquad N = N^{\uparrow} + N^{\downarrow}$$

Correlation (Jastrow)

$$J_1 = \sum_i^N \sum_I^{ions} u_1(|\mathbf{r}_i - \mathbf{r}_I|)$$

$$J_2 = \sum_{i \neq j}^N u_2(|\mathbf{r}_i - \mathbf{r}_j|)$$

Anti-symmetric function (Pauli principle)

$$D_i^{\uparrow} = \det \begin{vmatrix} \boxed{\begin{matrix} \phi_1(\mathbf{r}_1) \\ \vdots \\ \phi_{N^\uparrow}(\mathbf{r}_1) \end{matrix}} & \begin{matrix} \cdots \\ \vdots \\ \ldots \end{matrix} & \begin{matrix} \phi_1(\mathbf{r}_{N^\uparrow}) \\ \vdots \\ \phi_{N^\uparrow}(\mathbf{r}_{N^\uparrow}) \end{matrix} \end{vmatrix}$$

Single-particle orbitals

- Computational complexity per MC step
  - Evaluation $\{\phi\}$ — $\mathcal{O}(N^2 N_{spo})$
  - Determinant evaluation — $\mathcal{O}(N^3)$
  - Jastrow evaluation — $\mathcal{O}(N) - \mathcal{O}(N^3)$

Single-particle orbitals

- Linear combinations of basis functions

$$N_{spo} \propto N_b Op(\Phi)$$

$$\phi_i = \sum_{k}^{k=N_b} c_k^i \Phi_k$$

- Typically the solutions of simpler theories, i.e. $C's \, \& \{\Phi\}$ from Hartree-Fock or DFT calculations

- SPO can take various forms

SPO Type	N_b	$Op(\Phi)$	Memory Use
Molecular orbitals	$\mathcal{O}(N)$	Medium-High	Low
Plane waves	$\mathcal{O}(N)$	High	Medium
B-spline	Fixed	Low	High

Best solution for large-scale QMC on SMPs

GUIDES TO LABS

Setup

- Training accounts at forge.ncsa.illinois.edu
- Log on forge.ncsa.illinois.edu

  ssh -Y your-id@forge.ncsa.illinois.edu
- Setting environments and working directory (only once)

```
cp /uf/ac/jnkim/qmc/tcshrc_common ~/.tcshrc
cp /uf/ac/jnkim/qmc/modules ~/.modules
cd scratch-global
mkdir yourname
cd yourname
exit
```

- Throughout the lab, work on your working directory

```
cd scratch-global/yourname
```

QUICK REVIEW OF QMC

Basics of QMC

For *N*-electron system $\{\mathbf{R}\} = (\mathbf{r}_1, \cdots, \mathbf{r}_N)$

Many-body Hamiltonian $$\hat{H} = \sum_i \frac{1}{2m_e}\nabla^2 + \frac{1}{2}\sum_{i\neq j}\frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_i V_{ext}(\mathbf{r}_i)$$

Find the solution $\hat{H}|\Psi >= E_0|\Psi >$ & $\langle\Psi|\hat{H}|\Psi\rangle = E_0$

Many-body *trial* wavefunction $\Psi_T(\mathbf{R})$

$$E_T = \frac{\int d^{3N}\mathbf{R}\ \Psi_T^*(\mathbf{R})\hat{H}\Psi_T(\mathbf{R})}{\int d^{3N}\mathbf{R}\ |\Psi_T(\mathbf{R})|^2}, \quad E_T \geq E_0$$

QMC

$$\boxed{< E_T >= \frac{\sum_i^M w(\mathbf{R}_i)E_L(\mathbf{R}_i)}{\sum_i^M w(\mathbf{R}_i)}, \quad E_L = \frac{\hat{H}\Psi_T(\mathbf{R})}{\Psi_T(\mathbf{R})}}$$

Essentials of QMC

Note that

$$E_T = < E_T > |_{M\to\infty}, \quad E_0 \leftarrow E_T|_{\Psi_T\to\Psi}$$

QMC methods employ

- $\Psi_T(\mathbf{R})$, compact, fast to compute, and accurate
- Efficient stochastic sampling to generate large *M*

- Variational Monte Carlo (VMC)

Variational parameters

$$E_{VMC} = \min_{\alpha}\langle\Psi_T(\mathbf{R};\alpha)|\hat{H}|\Psi_T(\mathbf{R};\alpha)\rangle \qquad |\Psi_T|^2$$

- Diffusion Monte Carlo (DMC)

$$E_{DMC} = \langle\Phi_0|\hat{H}|\Psi_T\rangle, \quad \Phi_0 = \lim_{\beta\to\infty}\exp^{-\beta\hat{H}}\Psi_T \qquad \Phi_0\Psi_T$$

Efficiency of QMC

- QMC employs sampling to obtain

$$< E_T >= \frac{\sum_i^M w(\mathbf{R}_i) E_L(\mathbf{R}_i)}{\sum_i^M w(\mathbf{R}_i)}, \quad E_L = \frac{\hat{H}\Psi_T(\mathbf{R})}{\Psi_T(\mathbf{R})}$$

with an error bar $\delta = \frac{\sigma}{\sqrt{M}}$, $\boxed{\sigma^2 =< E_T^2 > - < E_T >^2}$

variance

- Minimize wall-clock time to reach a target error bar
- Efficiency of QMC simulations is high, when
  - Variance is small: $\sigma \to 0 \text{ as } \Psi_T \to \Psi$ (zero-variance)

    Physical insights & improved optimization
  - The rate of MC sample generation is high

    Parallelism, compact form of Ψ_T & optimized kernels

Diffusion Monte Carlo

```
for generation $= 1 \cdots N_{\text{MC}}$ do
  for walker $= 1 \cdots N_w$ do
    let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
    for particle $i = 1 \cdots N$ do
      set $\mathbf{r}_i' = \mathbf{r}_i + \delta$
      let $\mathbf{R}' = \{\mathbf{r}_1 \ldots \mathbf{r}_i' \ldots \mathbf{r}_N\}$
      ratio $\rho = \Psi_T(\mathbf{R}')/\Psi_T(\mathbf{R})$
      if $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted then
        update state of a walker
      end if
    end for{particle}
    Compute $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
    Reweight and branch walkers
    Update $E_T$
  end for{walker}
end for{generation}
```

Drift & Diffusion

Branch

Variation Monte Carlo

```
for generation = $1 \cdots N_{\text{MC}}$ do
  for walker = $1 \cdots N_w$ do
    let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
    for particle $i = 1 \cdots N$ do
      set $\mathbf{r}_i' = \mathbf{r}_i + \delta$
      let $\mathbf{R}' = \{\mathbf{r}_1 \ldots \mathbf{r}_i' \ldots \mathbf{r}_N\}$
      ratio $\rho = \Psi_T(\mathbf{R}')/\Psi_T(\mathbf{R})$
      if $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted then
        update state of a walker
      end if
    end for{particle}
    Compute $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
    No Branch
  end for{walker}
end for{generation}
```

useDrift="yes"

$$\delta = \tau \nabla_i \ln \Psi_T + \delta_{RN}$$

useDrift="no"

$$\delta = \delta_{RN}$$

Linear Optimization Method

Time-independent Schrödinger Eq.

$$\hat{H}|\Psi >= E_0|\Psi >$$

$$\langle\Psi|\hat{H}\,|\Psi\rangle = E_0$$

$$\Big\Downarrow$$

$$\overline{H}\Delta\mathbf{p} = E_{lin}\overline{S}\Delta\mathbf{p}$$

$$H_{ij} = \left\langle \frac{\overline{\Psi}_i}{\Psi_0}\frac{\hat{H}\overline{\Psi}_j}{\Psi_0} \right\rangle\Bigg|_{\Psi_0^2}$$

$$S_{ij} = \left\langle \frac{\overline{\Psi}_i}{\Psi_0}\frac{\overline{\Psi}_j}{\Psi_0} \right\rangle\Bigg|_{\Psi_0^2}$$

Expansion in basis of wave function derivatives

$$\Psi_{lin}(\mathbf{p},\mathbf{R}) = \Psi_0(\mathbf{R}) + \sum_i^{N_{opt}} \Delta p_i \boxed{\overline{\Psi}_i(\mathbf{R})}$$

$$\boxed{\frac{\partial\Psi}{\partial p_i} - \left\langle \frac{\partial\Psi}{\partial p_i}\middle|\Psi_0\right\rangle \Psi_0(\mathbf{R})}$$

* Umrigar et al, PRL 98, 110201 (2007)

Stabilizing Linear Optimization

- $\Delta\mathbf{p}$: lowest eingen vector but ill-conditioned eigenvalue, i.e., unbound E_{lin}
- Level Shift

$$H_{ij} = \left\langle \frac{\overline{\Psi}_i}{\Psi_0} \frac{\hat{H}\overline{\Psi}_j}{\Psi_0} + \delta_{ij} X_s \right\rangle \bigg|_{\Psi_0^2}, X_s > 0$$

- Rescaling for nonlinear parameters $\Delta\mathbf{p} \rightarrow \mathcal{R}\Delta\mathbf{p}$

$$\mathcal{R} = \frac{1}{1 + \frac{\xi D}{1-\xi+\xi\sqrt{1+D}}} \qquad D = \sum_{ij}^{nonlinear} \Delta p_i \Delta p_j S_{ij}$$

Toulouse and Umrigar, J. Chem. Phys. 126, 084102 (2007)

COMPUTATIONALLY SPEAKING

Characteristics of QMC

DMC pseudo code

```
for generation $= 1 \cdots N_{\mathrm{MC}}$ do
  for walker $= 1 \cdots N_w$ do
    let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
    for particle $i = 1 \cdots N$ do
      set $\mathbf{r}_i^{'} = \mathbf{r}_i + \delta$
      let $\mathbf{R}^{'} = \{\mathbf{r}_1 \ldots \mathbf{r}_i^{'} \ldots \mathbf{r}_N\}$
      ratio $\rho = \Psi_T(\mathbf{R}^{'})/\Psi_T(\mathbf{R})$
      if $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted then
        update state of a walker
      end if
    end for{particle}
    Compute $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
    Reweight and branch walkers
    Update $E_T$
  end for{walker}
end for{generation}
```

- Ample opportunity for parallelism
  - Configurations
  - K-point
  - Walker parallelization

Characteristics of QMC

DMC pseudo code

```
for generation $= 1 \cdots N_{\mathrm{MC}}$ do
  for walker $= 1 \cdots N_w$ do
    let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
    for particle $i = 1 \cdots N$ do
      set $\mathbf{r}_i^{'} = \mathbf{r}_i + \delta$
      let $\mathbf{R}^{'} = \{\mathbf{r}_1 \ldots \mathbf{r}_i^{'} \ldots \mathbf{r}_N\}$
      ratio $\rho = \Psi_T(\mathbf{R}^{'})/\Psi_T(\mathbf{R})$
      if $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted then
        update state of a walker
      end if
    end for{particle}
    Compute $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
    Reweight and branch walkers
    Update $E_T$
  end for{walker}
end for{generation}
```

- Ample opportunity for parallelism
  - Configurations
  - K-point
  - Walker parallelization
- Freedom in Ψ_T
  - Compute vs Memory
- Computationally demanding
  - Ratio, update & Local energy
  - Random access

MCC ORNL

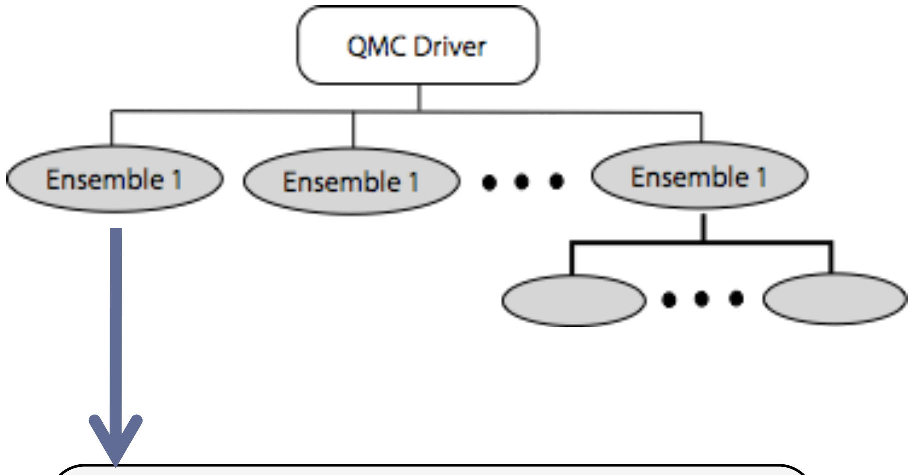
Characteristics of QMC

DMC pseudo code

```
for generation = $1 \cdots N_{\mathrm{MC}}$ do
  for walker = $1 \cdots N_w$ do
    let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
    for particle $i = 1 \cdots N$ do
      set $\mathbf{r}_i^{'} = \mathbf{r}_i + \delta$
      let $\mathbf{R}^{'} = \{\mathbf{r}_1 \ldots \mathbf{r}_i^{'} \ldots \mathbf{r}_N\}$
      ratio $\rho = \Psi_T(\mathbf{R}^{'})/\Psi_T(\mathbf{R})$
      if $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted then
        update state of a walker
      end if
    end for{particle}
    Compute $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
    Reweight and branch walkers
    Update $E_T$
  end for{walker}
end for{generation}
```

- Ample opportunity for parallelism
  - Configurations
  - K-point
  - Walker parallelization
- Freedom in Ψ_T
  - Compute vs Memory
- Computationally demanding
  - Ratio, update & Local energy
  - Random access
- Communication light but need to
  - Global sum
  - Load balance

Hierarchical Parallelization of QMC



```
for generation = $1 \cdots N_{\mathrm{MC}}$ do
    for walker = $1 \cdots N_w$ do
        A walker in cache
        Reweight and branch walkers
        Update $E_T$
    end for{walker}
end for{generation}
```

For a given *N*-electron system

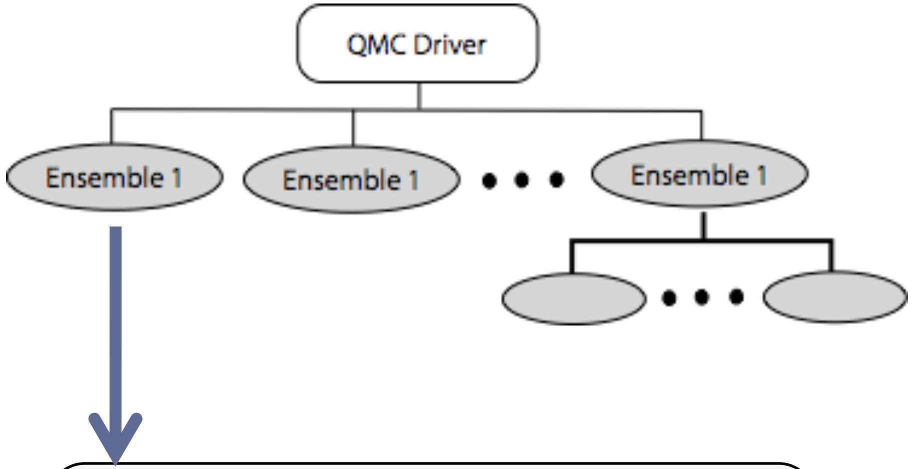
1 Multiple instances of correlated configurations: any

2 Multiple k-points : 1-100

Critical to remove finite-size effects

3 Walker parallelization: $N_w = 10^4 - 10^6$ Multi-core

Hierarchical Parallelization of QMC



```
for generation = $1 \cdots N_{\mathrm{MC}}$ do
    for walker = $1 \cdots N_w$ do

        Reweight and branch walkers
        Update $E_T$
    end for{walker}
end for{generation}
```

For a given *N*-electron system

1 Multiple instances of correlated configurations: any

2 Multiple k-points : 1-100

Critical to remove finite-size effects

3 Walker parallelization: $N_w = 10^4 - 10^6$

4 *N*-particle : $N - N^3$

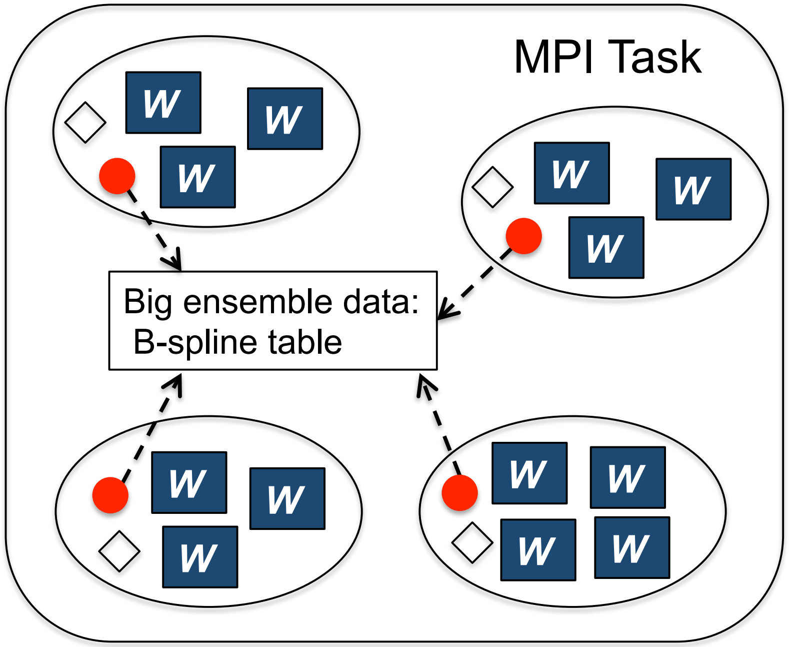
GPU

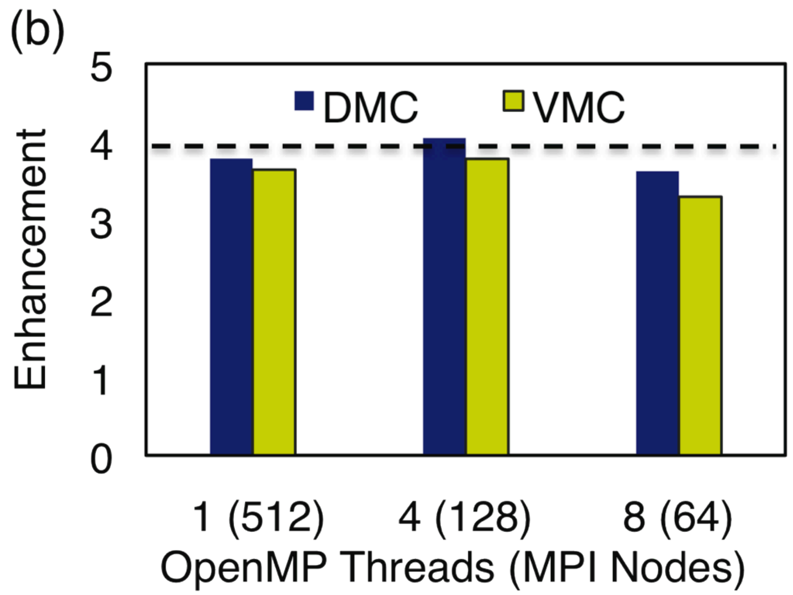
And, more parallelism can exposed

$$\Psi_T(\mathbf{R}) = \Pi_i \Psi_i, \hat{H} = \sum_i \hat{h}_i$$

Hybrid scheme on SMP

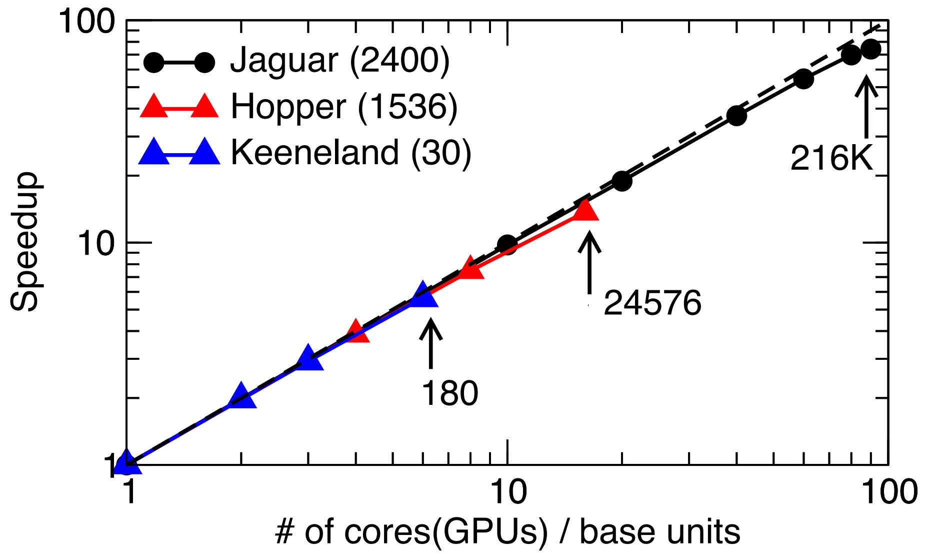
- Maximize performance and reduce the time-to-solution
    - MPI task per SMP, better per NUMA node
    - Multiple walkers per threads
    - Use all the hardware threads available

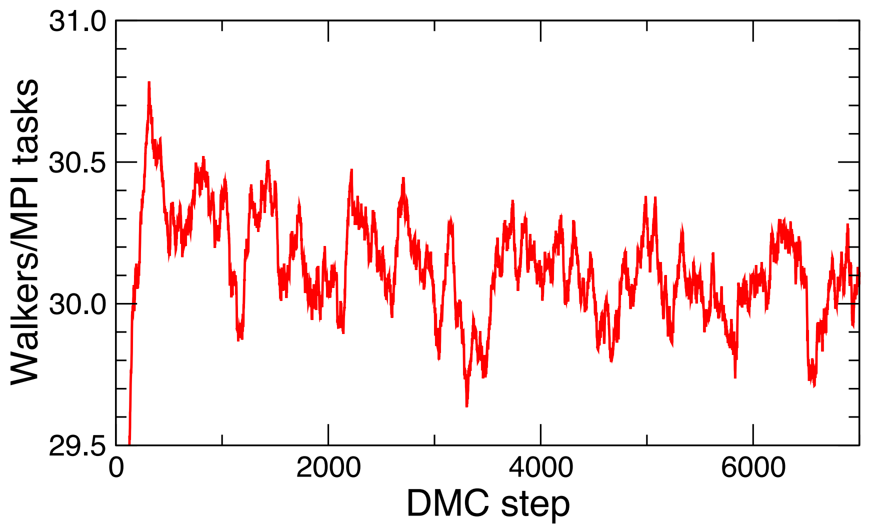




Performance of Hybrid QMC

- DMC scaling is almost perfect , > 90% efficiency
  - Limited by collectives for $E_T,\ N_p^w - < N^w >$
- Open/MPI hybrid helps more than memory footprint
  - Collectives scale O(P^2) *or O(P ln P)* for *P* tasks
  - Large average number of walkers per MPI task, thus small fluctuations : easy to balance walkers per node





MCC ORNL

INPUT & OUPUT OF QMCPACK

How to run QMCPACK

```
$export OMP_NUM_THREADS=4
$[mpirun –np 256] qmcapp input
```

- *input* is a valid XML file
- Using 4 OpenMP threads and 256 quad nodes
- May need other files, e.g., pseudopotential file

A sample input file

```xml
<?xml version="1.0"?>
<simulation>
  <project id="TITLE" series="0"/>
  <include href="PTCLXML"/>
  <include href="WFSXML"/>
  <include href="HAMXML"/>
  <loop max="5">
    <qmc method="cslinear" move="pbyp" checkpoint="-1" >
      <cost name="energy">                 0.05 </cost>
      <cost name="unreweightedvariance">0.00 </cost>
      <cost name="reweightedvariance">  0.95 </cost>
    </qmc>
  </loop>
  <qmc method="vmc" move="pbyp">
    <parameter name="timestep">1</parameter>
    <parameter name="samples">100000</parameter>
  </qmc>
  <qmc method="dmc" move="pbyp">
    <parameter name="timestep">0.02</parameter>
  </qmc>
</simulation>
```

Problem descriptions

- TITLE : project title
- PTCLXML : ions&els
- WFSXML : wavefunction
- HAMXML : hamiltonian

Optimization

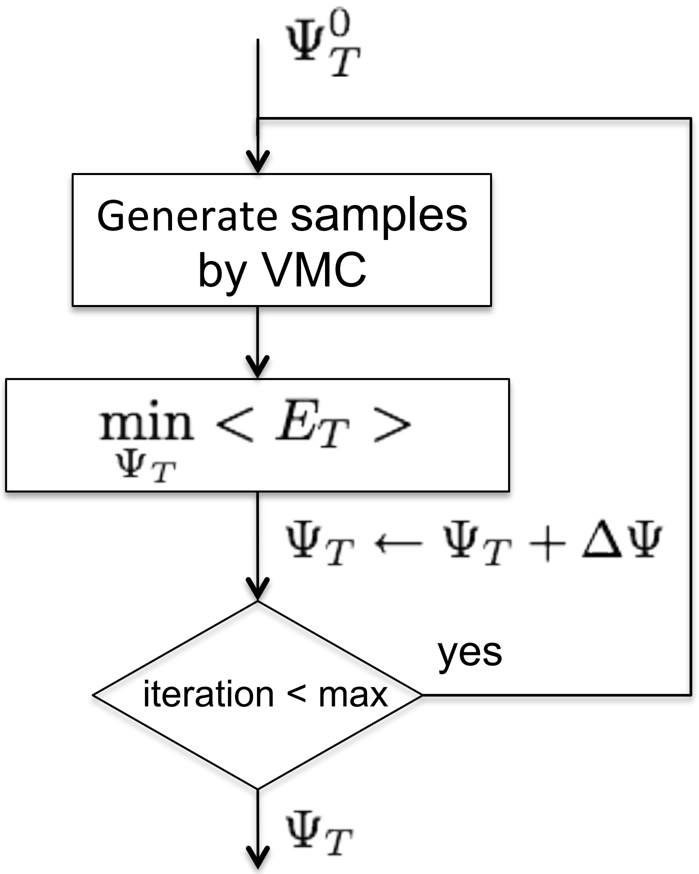
VMC

DMC

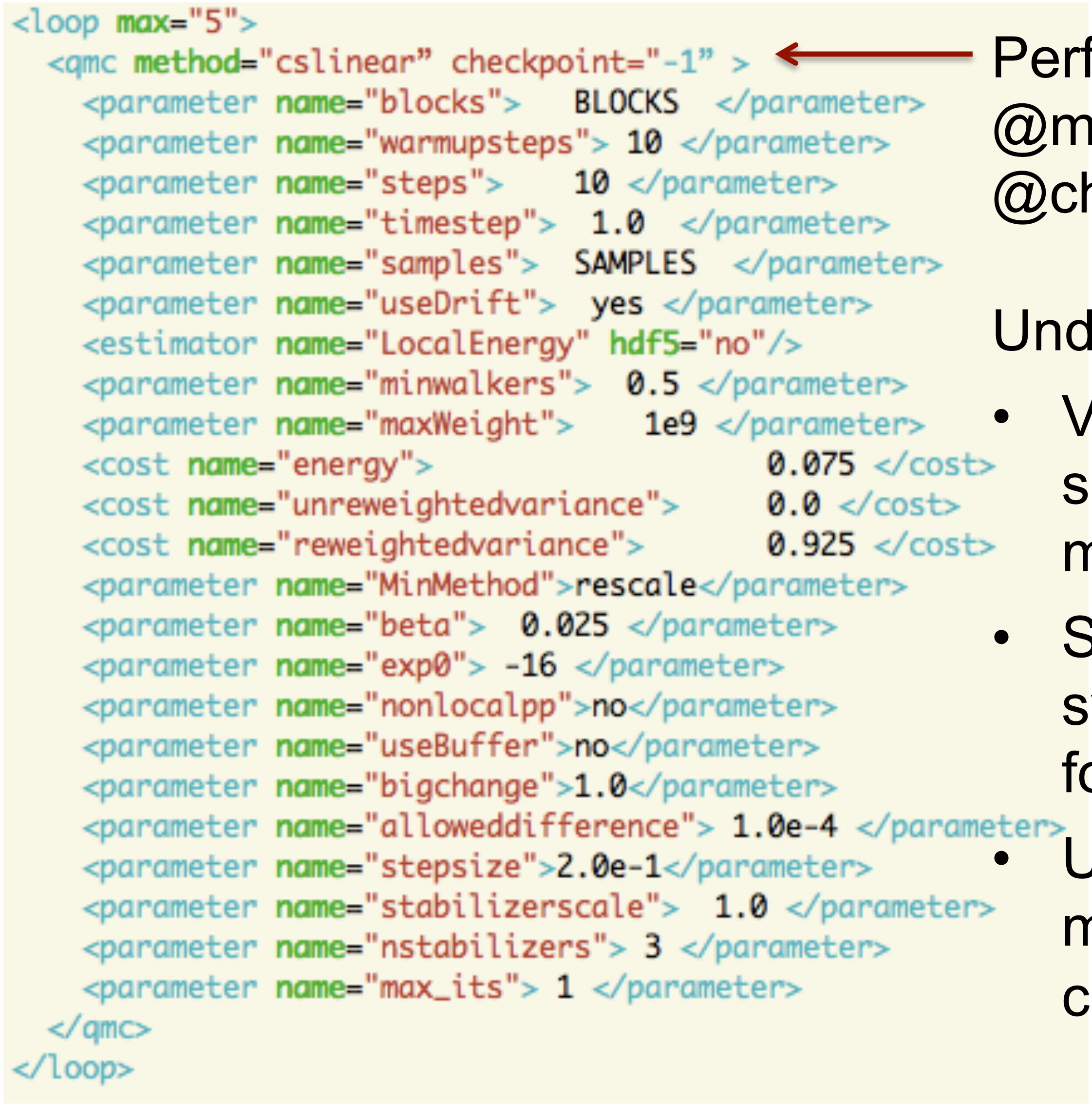
MCC ORNL

Optimization Loop: complete block

```
<loop max="5">
  <qmc method="cslinear" checkpoint="-1" >
    <parameter name="blocks">   BLOCKS  </parameter>
    <parameter name="warmupsteps"> 10 </parameter>
    <parameter name="steps">     10 </parameter>
    <parameter name="timestep">  1.0  </parameter>
    <parameter name="samples">  SAMPLES  </parameter>
    <parameter name="useDrift">  yes </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="minwalkers">  0.5 </parameter>
    <parameter name="maxWeight">    1e9 </parameter>
    <cost name="energy">                  0.075 </cost>
    <cost name="unreweightedvariance">      0.0 </cost>
    <cost name="reweightedvariance">        0.925 </cost>
    <parameter name="MinMethod">rescale</parameter>
    <parameter name="beta">  0.025 </parameter>
    <parameter name="exp0"> -16 </parameter>
    <parameter name="nonlocalpp">no</parameter>
    <parameter name="useBuffer">no</parameter>
    <parameter name="bigchange">1.0</parameter>
    <parameter name="alloweddifference"> 1.0e-4 </parameter>
    <parameter name="stepsize">2.0e-1</parameter>
    <parameter name="stabilizerscale">  1.0 </parameter>
    <parameter name="nstabilizers"> 3 </parameter>
    <parameter name="max_its"> 1 </parameter>
  </qmc>
</loop>
```



Optimization Loop: Summary



```
<loop max="5">
  <qmc method="cslinear” checkpoint="-1” >
    <parameter name="blocks">   BLOCKS  </parameter>
    <parameter name="warmupsteps"> 10 </parameter>
    <parameter name="steps">     10 </parameter>
    <parameter name="timestep">  1.0  </parameter>
    <parameter name="samples">  SAMPLES  </parameter>
    <parameter name="useDrift">  yes </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="minwalkers">  0.5 </parameter>
    <parameter name="maxWeight">    1e9 </parameter>
    <cost name="energy">                   0.075 </cost>
    <cost name="unreweightedvariance">     0.0 </cost>
    <cost name="reweightedvariance">       0.925 </cost>
    <parameter name="MinMethod">rescale</parameter>
    <parameter name="beta">  0.025 </parameter>
    <parameter name="exp0"> -16 </parameter>
    <parameter name="nonlocalpp">no</parameter>
    <parameter name="useBuffer">no</parameter>
    <parameter name="bigchange">1.0</parameter>
    <parameter name="alloweddifference"> 1.0e-4 </parameter>
    <parameter name="stepsize">2.0e-1</parameter>
    <parameter name="stabilizerscale">  1.0 </parameter>
    <parameter name="nstabilizers"> 3 </parameter>
    <parameter name="max_its"> 1 </parameter>
  </qmc>
</loop>
```

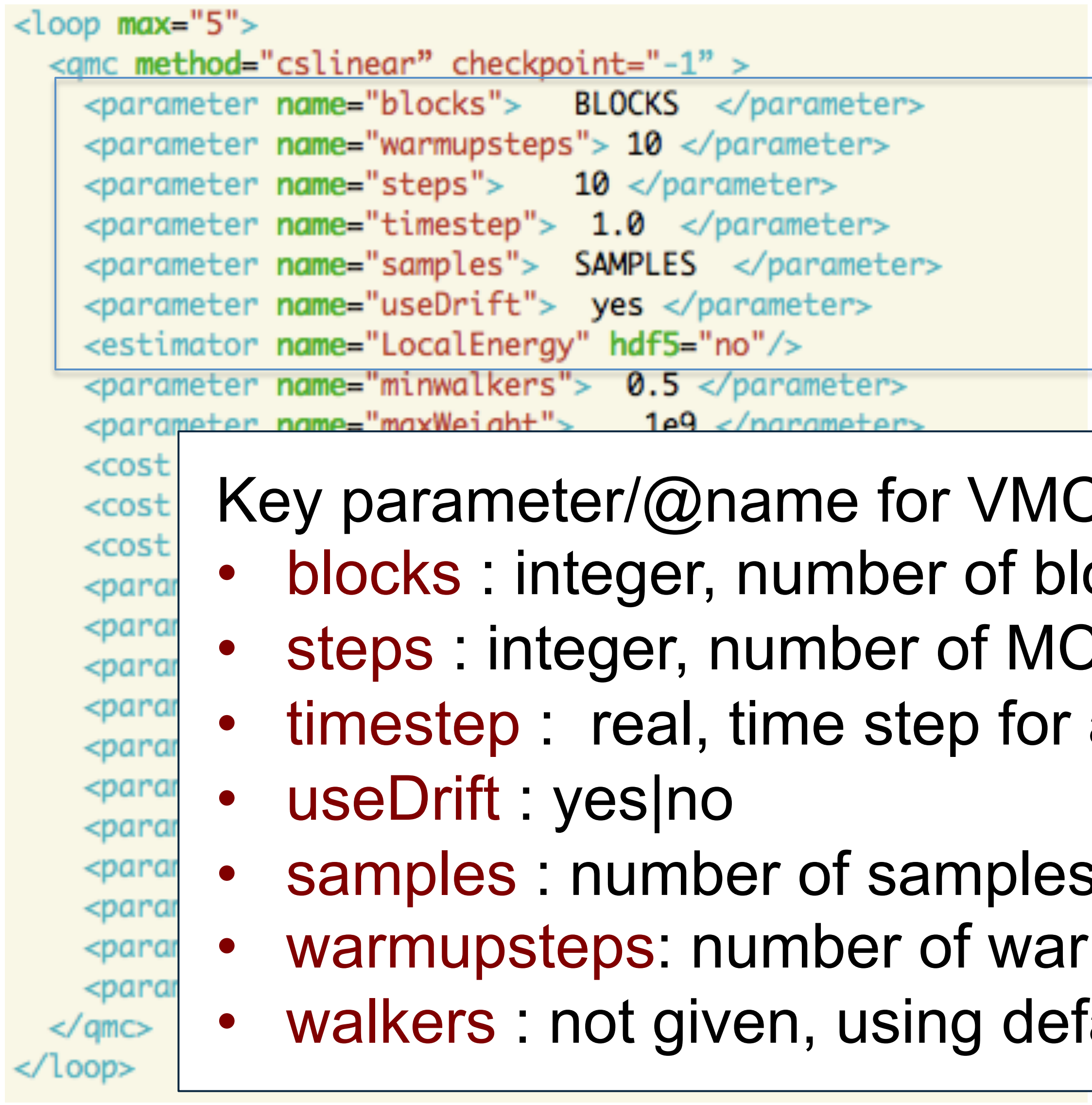
Perform an optimization
@method='cslinear'
@checkpoint='-1'

Under the hood

- VMC to generate uncorrelated samples & accumulate the matrix elements $\bar{H}\&\bar{S}$
- Solve a generalized non-symmetric eigenvalue problem for the few lowest eigen states
- Use correlated-sampling method to determine the "best" change in Ψ_T

MCPACK

Optimization: Sample Generation



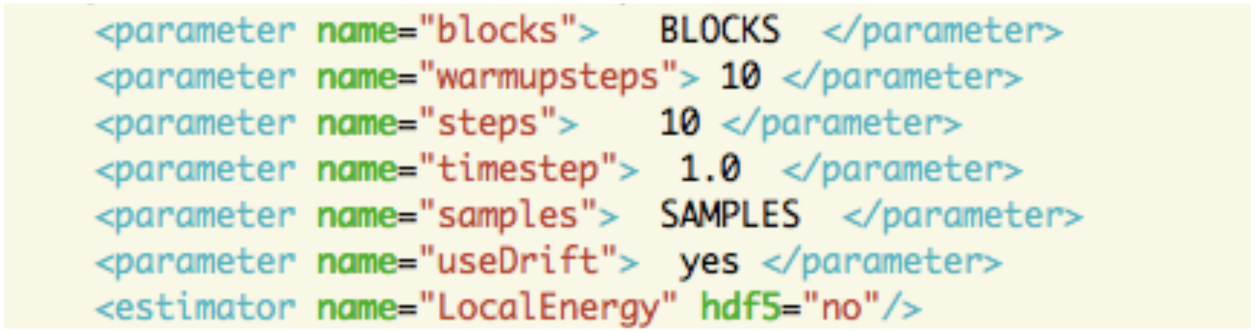
```
<loop max="5">
  <qmc method="cslinear" checkpoint="-1" >
    <parameter name="blocks">   BLOCKS  </parameter>
    <parameter name="warmupsteps"> 10 </parameter>
    <parameter name="steps">     10 </parameter>
    <parameter name="timestep">  1.0  </parameter>
    <parameter name="samples">  SAMPLES  </parameter>
    <parameter name="useDrift">  yes </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="minwalkers">  0.5 </parameter>
    <parameter name="maxWeight">     1e9 </parameter>
    <cost
    <cost
    <cost
    <para
    <para
    <para
    <para
    <para
    <para
    <para
    <para
    <para
    <para
    <para
  </qmc>
</loop>
```

Set VMC parameters

Key parameter/@name for VMC

- blocks : integer, number of blocks
- steps : integer, number of MC steps per block
- timestep : real, time step for a move
- useDrift : yes|no
- samples : number of samples used for optimization
- warmupsteps: number of warmup steps
- walkers : not given, using default=1 walker per thread

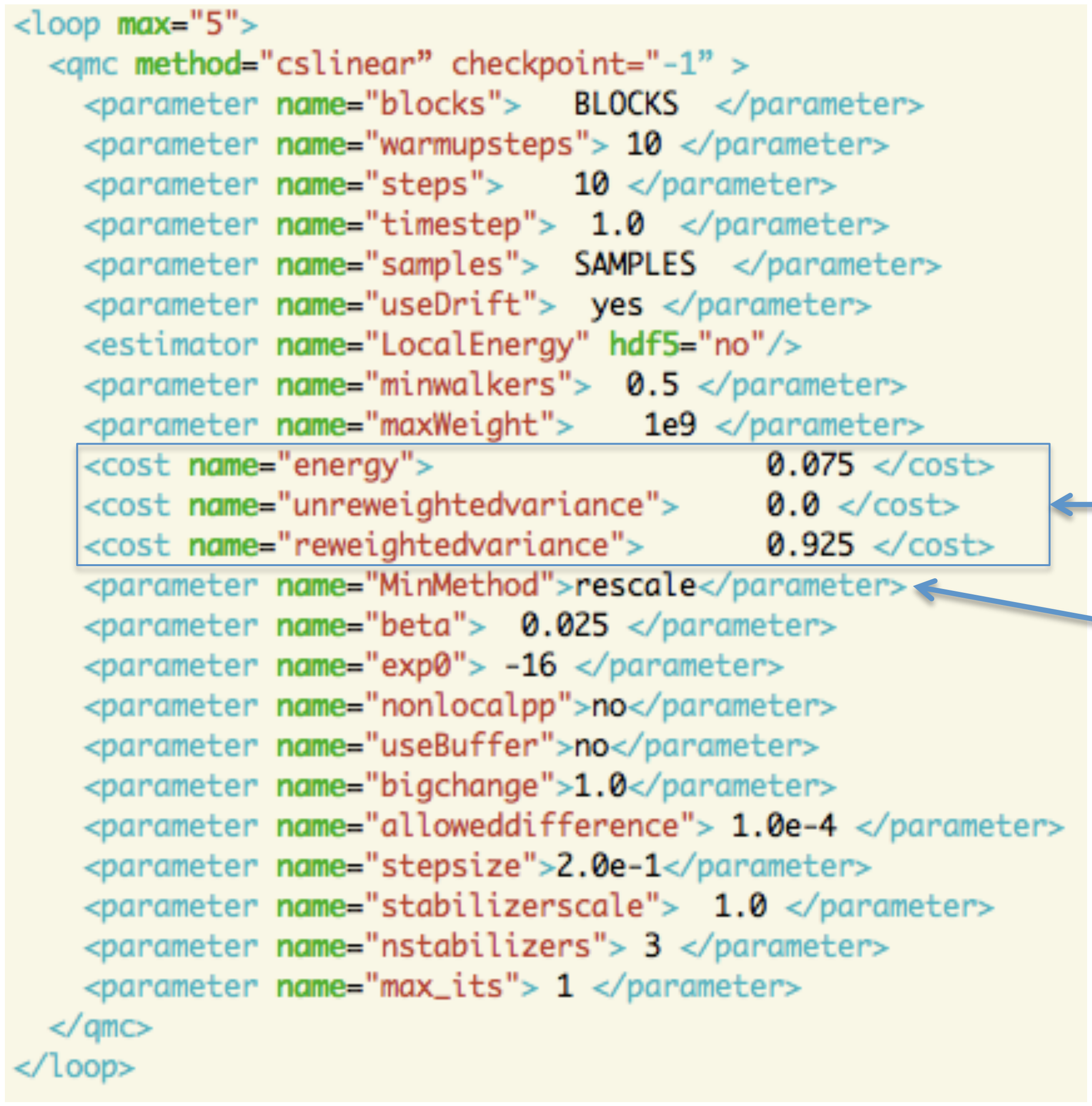
More on VMC parameters



```
<parameter name="blocks">    BLOCKS   </parameter>
<parameter name="warmupsteps"> 10 </parameter>
<parameter name="steps">     10 </parameter>
<parameter name="timestep">  1.0  </parameter>
<parameter name="samples">   SAMPLES   </parameter>
<parameter name="useDrift">  yes </parameter>
<estimator name="LocalEnergy" hdf5="no"/>
```

- VMC is used to generate uncorrelated samples.
- Replace BLOCKS and SAMPLES by integers!
- Total MC configurations=BLOCKS*STEPS*MPI*THREADS
  - MPI*THREADS=total number of cores on multi-core
- Data during warmupsteps are discarded
- timestep~1 Hartree^{-1} for pseudopotentials
  - Check the acceptance rate: target 30-50%
- samples are internally stored during a VMC run
  - Use as many samples as possible $> 100N_{opt}^2$

Primary Optimization Parameters



```
<loop max="5">
  <qmc method="cslinear” checkpoint="-1” >
    <parameter name="blocks">   BLOCKS  </parameter>
    <parameter name="warmupsteps"> 10 </parameter>
    <parameter name="steps">    10 </parameter>
    <parameter name="timestep">  1.0  </parameter>
    <parameter name="samples">  SAMPLES  </parameter>
    <parameter name="useDrift">  yes </parameter>
    <estimator name="LocalEnergy" hdf5="no"/>
    <parameter name="minwalkers">  0.5 </parameter>
    <parameter name="maxWeight">    1e9 </parameter>
    <cost name="energy">                  0.075 </cost>
    <cost name="unreweightedvariance">     0.0 </cost>
    <cost name="reweightedvariance">      0.925 </cost>
    <parameter name="MinMethod">rescale</parameter>
    <parameter name="beta">  0.025 </parameter>
    <parameter name="exp0"> -16 </parameter>
    <parameter name="nonlocalpp">no</parameter>
    <parameter name="useBuffer">no</parameter>
    <parameter name="bigchange">1.0</parameter>
    <parameter name="alloweddifference"> 1.0e-4 </parameter>
    <parameter name="stepsize">2.0e-1</parameter>
    <parameter name="stabilizerscale">  1.0 </parameter>
    <parameter name="nstabilizers"> 3 </parameter>
    <parameter name="max_its"> 1 </parameter>
  </qmc>
</loop>
```

Cost (object) function

linear combination of energy and variance

MinMethod=(rescale,quartic,linemin)

Methods to choose "best" $\Delta\mathbf{p}$

- Find the optimal X_s
- Solve general eigenvalue problem to get $\Delta\mathbf{p}$
- Rescale $\Delta\mathbf{p} \rightarrow \mathcal{R}\Delta\mathbf{p}$
- How much along $\Delta\mathbf{p}$

parameter/@MinMethod

- rescale : Original Linear Optimization Stabilization
- quartic : line-minimization by fitting a 4th-order polynomial
- linemin : line-minimization by bracketing

DMC

```
<qmc method="vmc" multiple="no" warp="no" checkpoint="100" move="pbyp" gpu="yes">
  <estimator name="LocalEnergy" hdf5="no"/>
  <parameter name="useDrift">yes</parameter>
  <parameter name="blocks">VMCBLOCKS</parameter>
  <parameter name="steps">VMCSTEPS</parameter>
  <parameter name="walkers">1</parameter>
  <parameter name="samples">SAMPLES</parameter>
  <parameter name="warmupsteps">10</parameter>
  <parameter name="timestep">2.0</parameter>
</qmc>
<qmc method="dmc" multiple="no" warp="no" checkpoint="100"  move="pbyp" gpu="yes">
  <estimator name="LocalEnergy" hdf5="no"/>
  <parameter name="nonlocalmoves"> yes </parameter>
  <parameter name="warmupsteps">100</parameter>
  <parameter name="blocks">DMCBLOCKS</parameter>
  <parameter name="steps">10</parameter>
  <parameter name="timestep">0.02</parameter>
</qmc>
```

VMC

DMC

- Recommend a short VMC before DMC to generate the initial population of SAMPLES walkers
  - Speedup DMC equilibration

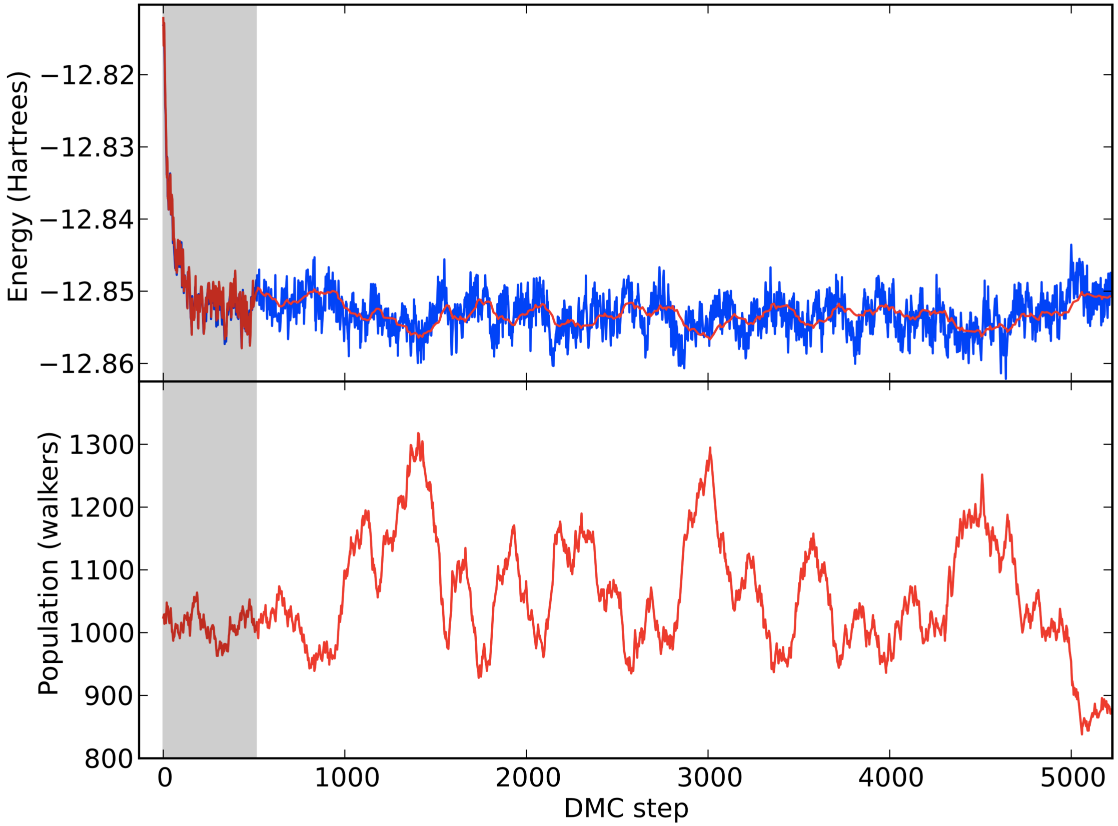
Output of QMCPACK

- Each qmc section has its own set of outputs
  - `TITLE.s###.scalar.dat` : block averages
  - `TITLE.s###.dmc.dat` : averages per DMC step
  - `TITLE.s###.stat.h5` : block avegrages in HDF5
  - `TITLE.s###.config.h5` : configuration for restart
- Use scripts in utils

```
energy.pl TITLE.s000.scalar.dat 0
```

Plotting averages

- `*.scalar.dat` and `*.dmc.dat` are simple tables with header lines starting with #



BUILDING QMCPACK

Requirements

- Must have
  - C++ : latest GNU and Intel compilers
  - cmake : build utility
  - BLAS/LAPACK : MKL, ACML, ESSL, netlib
  - libxml2 : XML I/O library
  - HDF5 : Portable hierarchical data format
  - boost
- Should have
  - einspline: 3D bspline library
  - FFTW : FFT library

Quick Start

```
1) $svn co http://qmcpack.googlecode.com/svn/trunk qmcpack
2) $export HDF5_HOME=/usr/local/hdf/hdf5/v188
3) $cd qmcpack/build
4) $cmake ..
5) $make –j8
```

1) Download QMCPACK
2) Set environment variables: compilers, LIBRARY paths
3) Change to a build directory
4) Run `cmake` to configure `Makefiles`
5) Run `make` with 8 threads

Setting Library Paths

- Pass –`DXYZ_HOME=`*`where-is-XYZ`*
- XYZ =(LIBXML2, HDF5, FFTW,EINSPLINE)

  For each XYZ library, `cmake` searches

  – *`where-is-XYZ`*`/include`

  – *`where-is-XYZ`*`/lib`

  in addition to the standard include and lib paths
- Boost library : BOOST_HOME
  - Only need header files, do not need to build
  - Search *`where-is-boost`*`/boost/config.hpp`

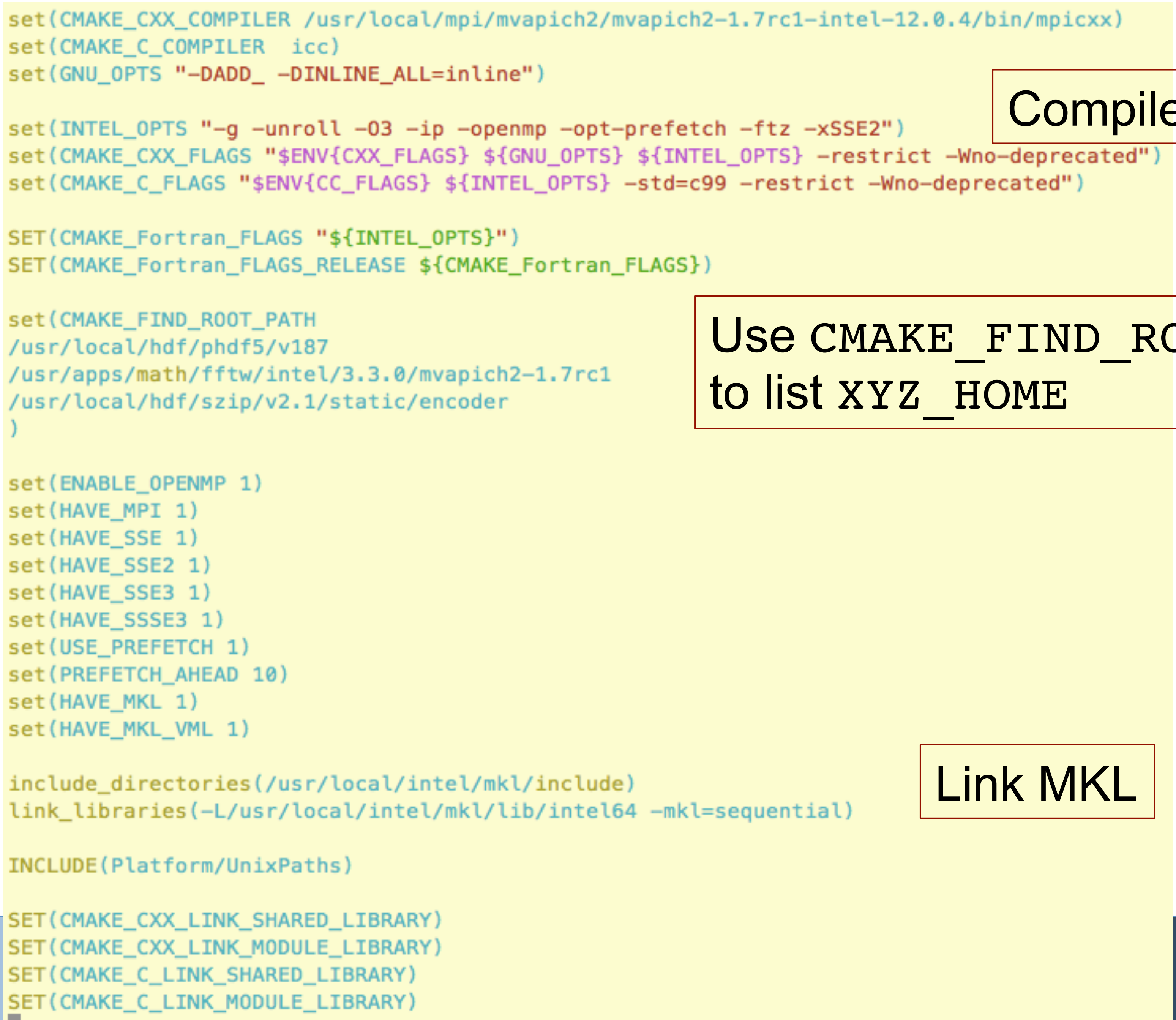
Customizing build with a toolchain

- Choose a toolchain file in `qmcpack/config`
- Example on forge@ncsa (Linux)

```
$cd qmcpack/build
$cp ../config/ForgeMvapich.cmake mytool.cmake
$cmake -DCMAKE_TOOLCHAIN_FILE=mytool.cmake ..
$cmake -DCMAKE_TOOLCHAIN_FILE=mytool.cmake ..
$make –j8
```

- Edit `mytool.cmake` to keep up with the compiler and library updates

ForgeMvapich.cmake



```
set(CMAKE_CXX_COMPILER /usr/local/mpi/mvapich2/mvapich2-1.7rc1-intel-12.0.4/bin/mpicxx)
set(CMAKE_C_COMPILER  icc)
set(GNU_OPTS "-DADD_ -DINLINE_ALL=inline")

set(INTEL_OPTS "-g -unroll -O3 -ip -openmp -opt-prefetch -ftz -xSSE2")
set(CMAKE_CXX_FLAGS "$ENV{CXX_FLAGS} ${GNU_OPTS} ${INTEL_OPTS} -restrict -Wno-deprecated")
set(CMAKE_C_FLAGS "$ENV{CC_FLAGS} ${INTEL_OPTS} -std=c99 -restrict -Wno-deprecated")

SET(CMAKE_Fortran_FLAGS "${INTEL_OPTS}")
SET(CMAKE_Fortran_FLAGS_RELEASE ${CMAKE_Fortran_FLAGS})

set(CMAKE_FIND_ROOT_PATH
/usr/local/hdf/phdf5/v187
/usr/apps/math/fftw/intel/3.3.0/mvapich2-1.7rc1
/usr/local/hdf/szip/v2.1/static/encoder
)

set(ENABLE_OPENMP 1)
set(HAVE_MPI 1)
set(HAVE_SSE 1)
set(HAVE_SSE2 1)
set(HAVE_SSE3 1)
set(HAVE_SSSE3 1)
set(USE_PREFETCH 1)
set(PREFETCH_AHEAD 10)
set(HAVE_MKL 1)
set(HAVE_MKL_VML 1)

include_directories(/usr/local/intel/mkl/include)
link_libraries(-L/usr/local/intel/mkl/lib/intel64 -mkl=sequential)

INCLUDE(Platform/UnixPaths)

SET(CMAKE_CXX_LINK_SHARED_LIBRARY)
SET(CMAKE_CXX_LINK_MODULE_LIBRARY)
SET(CMAKE_C_LINK_SHARED_LIBRARY)
SET(CMAKE_C_LINK_MODULE_LIBRARY)
```

Compiler options

Use `CMAKE_FIND_ROOT_PATH` to list `XYZ_HOME`

Link MKL