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

QMC Endstation

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

Special thanks to

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

*http://qmcpack.cmscc.org/
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

<table>
<thead>
<tr>
<th>QMC Methods</th>
<th>Many-body Wavefunction</th>
<th>Many-body Hamiltonian</th>
</tr>
</thead>
<tbody>
<tr>
<td>• VMC, DMC, RMC</td>
<td>• 1,2, 3-body &amp; k-space Jastrow</td>
<td>• Kinetic operator</td>
</tr>
<tr>
<td>• Correlated sampling in VMC,RMC</td>
<td>• Single Slater-Jastrow</td>
<td>• Coulomb potential</td>
</tr>
<tr>
<td>• Optimization based on linear method: energy and variance minimization</td>
<td>• Multi Slater-Jastrow</td>
<td>• Pseudopotentials</td>
</tr>
<tr>
<td></td>
<td>• Backflow</td>
<td>• Multiple forms of He</td>
</tr>
<tr>
<td></td>
<td>• AGP in LCAO</td>
<td>• Model Hamiltonian on 3D grid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming and platforms</th>
<th>Single-particle orbitals</th>
<th>Finite-size corrections</th>
</tr>
</thead>
<tbody>
<tr>
<td>• C++ and CUDA</td>
<td>• Plane Wave</td>
<td>• Twist averaging</td>
</tr>
<tr>
<td>• Object-oriented, generic programming and design patterns</td>
<td>• Linear combination of atomic orbitals (LCAO): Gaussian, Slater, numerical and mixed basis</td>
<td>• Chiesa correction</td>
</tr>
<tr>
<td>• (OpenMP,CUDA)/MPI parallelization</td>
<td>• B-spline (einspline)</td>
<td>• Modified potential correction</td>
</tr>
<tr>
<td>• Ported on clusters of SMP: x86, IBM Power and Blue Gene, etc</td>
<td>• LAPW with B-spline</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
QMC Workflow

- **Initial guess** $\Psi_T^0$
  - Compact, easy to evaluate, but close to true $\Psi$
    - Single-particle orbitals \{\phi\}
      - e.g., KS or HF solution

\[
\Psi_T(R) = J(\{\alpha\}) \sum C_i D_i^\dagger(\phi) D_i^\dagger(\phi)
\]

- **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,

\[
\begin{align*}
\text{let } R &= \{ r_1 \ldots r_N \} \\
\text{for } \text{particle } i = 1 \ldots N \text{ do} \\
& \quad \text{set } r'_i = r_i + \delta \\
& \quad \text{let } R' = \{ r_1 \ldots r'_i \ldots r_N \} \\
& \quad \text{ratio } \rho = \frac{\Psi_T(R')}{\Psi_T(R)} \\
& \quad \text{if } r \rightarrow r' \text{ is accepted then} \\
& \quad \quad \text{update state of a walker} \\
& \quad \text{end if} \\
\text{end for}\{\text{particle}\} \ \\
\text{Compute } E_L = \hat{H} \Psi_T(R) / \Psi_T(R)
\end{align*}
\]

All about $\Psi_T$

\[
\begin{align*}
\delta &= r + \tau \nabla_i \ln \Psi_T \\
\Psi_T(r_1 \ldots r'_i \ldots r_N) &= \frac{\Psi_T(r_1 \ldots r_i \ldots r_N)}{\Psi_T(r_1 \ldots r'_i \ldots r_N)} \\
\Psi_T &\leftarrow \Psi_T(r_1 \ldots r'_i \ldots r_N) \\
f(\{R\}, \nabla \ln \Psi_T, \nabla^2 \ln \Psi_T)
\end{align*}
\]

Use $\Psi_T = \prod_i \Psi_i \quad \rightarrow \quad \ln \Psi_T = \sum_i \ln \Psi_i$
Slater-Jastrow for Electrons

\[ \Psi_T(R) = e^{J_1 + J_2 + \cdots} \sum C_i D_{i}^\uparrow(\phi) D_{i}^\downarrow(\phi) \quad N = N^\uparrow + N^\downarrow \]

Correlation (Jastrow)

\[ J_1 = \sum_{i} \sum_{I} u_1(|r_i - r_I|) \]
\[ J_2 = \sum_{i \neq j} u_2(|r_i - r_j|) \]

Anti-symmetric function (Pauli principle)

\[ D_i^\uparrow = \text{det} \begin{bmatrix} \phi_1(r_1) & \cdots & \phi_1(r_{N^\uparrow}) \\ \vdots & \ddots & \vdots \\ \phi_{N^\uparrow}(r_1) & \cdots & \phi_{N^\uparrow}(r_{N^\uparrow}) \end{bmatrix} \]

Single-particle orbitals

• Computational complexity per MC step
  • Evaluation \{\phi\}
  • Determinant evaluation
  • Jastrow evaluation

\[ \mathcal{O}(N^2 N_{spo}) \]
\[ \mathcal{O}(N^3) \]
\[ \mathcal{O}(N) - \mathcal{O}(N^3) \]
Single-particle orbitals

• Linear combinations of basis functions

\[ N_{spo} \propto N_b Op(\Phi) \]

• Typically the solutions of simpler theories, i.e. from Hartree-Fock or DFT calculations

\[ \phi_i = \sum_{k=1}^{N_b} c_k^i \Phi_k \]

• SPO can take various forms

<table>
<thead>
<tr>
<th>SPO Type</th>
<th>( N_b )</th>
<th>( Op(\Phi) )</th>
<th>Memory Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molecular orbitals</td>
<td>( O(N) )</td>
<td>Medium-High</td>
<td>Low</td>
</tr>
<tr>
<td>Plane waves</td>
<td>( O(N) )</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>B-spline</td>
<td>Fixed</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

Best solution for large-scale QMC on SMPs
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

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

Find the solution

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

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

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

Note that

\[ E_T = \langle E_T \rangle_{M \to \infty}, \quad E_0 \leftarrow E_T |_{\Psi_T \to \Psi} \]

QMC methods employ

- \( \Psi_T(R) \), compact, fast to compute, and accurate
- Efficient stochastic sampling to generate large \( M \)
- Variational Monte Carlo (VMC)
  \[ E_{VMC} = \min_{\alpha} \langle \Psi_T(R; \alpha) | \hat{H} | \Psi_T(R; \alpha) \rangle | \Psi_T |^2 \]
  
  \( \Psi_T \) is the trial wave function, \( \beta \) is the inverse temperature.

- Diffusion Monte Carlo (DMC)
  \[ E_{DMC} = \langle \Phi_0 | \hat{H} | \Psi_T \rangle, \quad \Phi_0 = \lim_{\beta \to \infty} \exp^{-\beta \hat{H}} \Psi_T \]
  
  \( \Phi_0 \) is the target wave function.
Efficiency of QMC

- QMC employs sampling to obtain
  \[< E_T > = \frac{\sum_i^M w(R_i)E_L(R_i)}{\sum_i^M w(R_i)}, \quad E_L = \frac{\hat{H}\Psi_T(R)}{\Psi_T(R)}\]
  with an error bar
  \[\delta = \frac{\sigma}{\sqrt{M}}, \quad \sigma^2 = < E_T^2 > - < E_T >^2\]

- Minimize wall-clock time to reach a target error bar

- Efficiency of QMC simulations is high, when
  - Variance is small: \(\sigma \rightarrow 0\) as \(\Psi_T \rightarrow \Psi\) (zero-variance)
    - Physical insights & improved optimization
  - The rate of MC sample generation is high
    - Parallelism, compact form of \(\Psi_T\) & optimized kernels
**Diffusion Monte Carlo**

for generation = 1 \cdots N_{MC} do \\
for walker = 1 \cdots N_{w} do \\
  let R = \{r_1 \ldots r_N\} \\
  for particle i = 1 \cdots N do \\
    set r_i' = r_i + \delta \\
    let R' = \{r_1 \ldots r_i' \ldots r_N\} \\
    ratio \ \rho = \Psi_T(R')/\Psi_T(R) \\
    if r \rightarrow r' is accepted then \\
      update state of a walker \\
    end if \\
  end for \\
  Compute \ \hat{E}_L = \hat{H}\Psi_T(R)/\Psi_T(R) \\
  Reweight and branch walkers \\
  Update \ E_T \\
end for \\
end for \\

Drift & Diffusion

Branch
Variation Monte Carlo

\[
\text{for generation } = 1 \cdots N_{\text{MC}} \text{ do}
\]
\[
\text{for walker } = 1 \cdots N_{\text{w}} \text{ do}
\]
\[
\text{let } R = \{r_1 \ldots r_N\}
\]
\[
\text{for particle } i = 1 \cdots N \text{ do}
\]
\[
\text{set } r_i' = r_i + \delta
\]
\[
\text{let } R' = \{r_1 \ldots r_i' \ldots r_N\}
\]
\[
\text{ratio } \rho = \Psi_T(R')/\Psi_T(R)
\]
\[
\text{if } r \rightarrow r' \text{ is accepted then}
\]
\[
\text{update state of a walker}
\]
\[
\text{end if}
\]
\[
\text{end for}\{\text{particle}\}
\]
\[
\text{Compute } E_L = \hat{H}\Psi_T(R)/\Psi_T(R)
\]
\[
\text{end for}\{\text{walker}\}
\]
\[
\text{end for}\{\text{generation}\}
\]

\[
\text{useDrift=“yes”}
\]
\[
\delta = \tau \nabla_i \ln \Psi_T + \delta_{RN}
\]

\[
\text{useDrift=“no”}
\]
\[
\delta = \delta_{RN}
\]

No Branch
Linear Optimization Method

Time-independent Schrödinger Eq.

\[ \hat{H} |\Psi\rangle = E_0 |\Psi\rangle \]

\[ \langle \Psi | \hat{H} | \Psi \rangle = E_0 \]

\[ \overline{H} \Delta p = E_{lin} \overline{S} \Delta p \]

\[ H_{ij} = \left\langle \frac{\overline{\Psi}_i \hat{H} \overline{\Psi}_j}{\overline{\Psi}_0 \overline{\Psi}_0} \right\rangle \bigg|_{\overline{\Psi}_0^2} \]

\[ S_{ij} = \left\langle \frac{\overline{\Psi}_i \overline{\Psi}_j}{\overline{\Psi}_0 \overline{\Psi}_0} \right\rangle \bigg|_{\overline{\Psi}_0^2} \]

Expansion in basis of wave function derivatives

\[ \Psi_{lin}(p, R) = \Psi_0(R) \]

\[ + \sum_{i} \Delta p_i \overline{\Psi}_i(R) \]

\[ \frac{\partial \Psi}{\partial p_i} - \left\langle \frac{\partial \Psi}{\partial p_i} \right\rangle \Psi_0 \bigg| \Psi_0(R) \]

* Umrigar et al, PRL 98, 110201 (2007)
Stabilizing Linear Optimization

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

\[
H_{ij} = \left\langle \frac{\Psi_i \hat{H} \Psi_j}{\Psi_0 \Psi_0} + \delta_{ij} X_s \right\rangle \bigg|_{\Psi_0^2}, X_s > 0
\]

- Rescaling for nonlinear parameters $\Delta p \rightarrow R \Delta p$

\[
R = \frac{1}{1 + \frac{\xi D}{1-\xi+\xi \sqrt{1+D}}}
\]

\[
D = \sum_{ij} \Delta p_i \Delta p_j S_{ij}
\]

COMPUTATIONALLY SPEAKING
Characteristics of QMC

DMC pseudo code
for generation = 1 · · · N_{\text{MC}} do
  for walker = 1 · · · N_{\text{w}} do
    let \( \mathbf{R} = \{ \mathbf{r}_1 \ldots \mathbf{r}_N \} \)
    for particle \( i = 1 \ldots N \) 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
  end for
Compute \( E_L = \hat{H} \Psi_T(\mathbf{R})/\Psi_T(\mathbf{R}) \)
Reweight and branch walkers
Update \( E_T \)
end for
end for

- Ample opportunity for parallelism
  - Configurations
  - K-point
  - Walker parallelization
Characteristics of QMC

DMC pseudo code

\[
\text{for generation } = 1 \cdots N_{\text{MC}} \text{ do}
\]

\[
\text{for walker } = 1 \cdots N_w \text{ do}
\]

let \( R = \{r_1 \ldots r_N\} \)

\[
\text{for particle } i = 1 \cdots N \text{ do}
\]

set \( r_i' = r_i + \delta \)

let \( R' = \{r_1 \ldots r_i' \ldots r_N\} \)

\[
\text{ratio } \rho = \Psi_T(R')/\Psi_T(R)
\]

\[
\text{if } r \rightarrow r' \text{ is accepted then}
\]

update state of a walker

\[
\text{end if}
\]

\[
\text{end for\{particle\}}
\]

Compute \( E_L = \hat{H}\Psi_T(R)/\Psi_T(R) \)

Reweight and branch walkers

Update \( E_T \)

\[
\text{end for\{walker\}}
\]

\[
\text{end for\{generation\}}
\]

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

- Freedom in \( \Psi_T \)
  - Compute vs Memory

- Computationally demanding
  - Ratio, update & Local energy
  - Random access
Characteristics of QMC

DMC pseudo code

\[
\text{for generation } = 1 \cdots N_{MC} \text{ do}
\]
\[
\text{for walker } = 1 \cdots N_w \text{ do}
\]
\[
\text{let } R = \{r_1 \ldots r_N\}
\]
\[
\text{for particle } i = 1 \cdots N \text{ do}
\]
\[
\text{set } r'_i = r_i + \delta
\]
\[
\text{let } R' = \{r_1 \ldots r'_i \ldots r_N\}
\]
\[
\text{ratio } \rho = \Psi_T(R')/\Psi_T(R)
\]
\[
\text{if } r \rightarrow r' \text{ is accepted then}
\]
\[
\text{update state of a walker}
\]
\[
\text{end if}
\]
\[
\text{end for} \{\text{particle}\}
\]
\[
\text{Compute } E_L = \hat{H}\Psi_T(R)/\Psi_T(R)
\]
\[
\text{Reweight and branch walkers}
\]
\[
\text{Update } E_T
\]
\[
\text{end for} \{\text{walker}\}
\]
\[
\text{end for} \{\text{generation}\}
\]

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

- Freedom in $\Psi_T$
  - 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 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

\[
\text{for generation} = 1 \cdots N_{MC} \text{ do}
\]
\[
\text{for walker} = 1 \cdots N_w \text{ do}
\]

A walker in cache

Reweight and branch walkers
Update $E_T$

end for{walker}
end for{generation}
Hierarchical Parallelization of QMC

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 be exposed
\[ \Psi_T(R) = \Pi_i \Psi_i, \hat{H} = \sum_i \hat{h}_i \]
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

Big ensemble data: B-spline table
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
INPUT & OUTPUT OF QMCPACK
How to run QMCPACK

```bash
$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
Problem descriptions

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

Optimization

VMC

DMC
Optimization Loop: complete block

```xml
<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="unreweightededvariance">0.0 </cost>
    <cost name="reweightededvariance">0.925 </cost>
    <parameter name="MinMethod">rescale</parameter>
    <parameter name="beta">0.025 </parameter>
    <parameter name="exp0">-16 </parameter>
    <parameter name="nonlocalbpp">no</parameter>
    <parameter name="useBuffer">no</parameter>
    <parameter name="bigchange">1.0</parameter>
    <parameter name="allowedddifference">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>
```

Generate samples by VMC

\[ \min < E_T > \]

\[ \Psi_T \rightarrow \Psi_T + \Delta \Psi \]

iteration < max

yes
Optimization Loop: Summary

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

Under the hood

• VMC to generate uncorrelated samples & accumulate the matrix elements $\hat{H} & \hat{S}$
• Solve a generalized non-symmetric eigenvalue problem for the few lowest eigen states
• Use correlated-sampling method to determine the “best” change in $\Psi_T$
Optimization: Sample Generation

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

Set VMC parameters
More on VMC parameters

- 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

Cost (object) function:
linear combination of energy and variance

MinMethod = (rescale, quartic, linemin)
Methods to choose “best” $\Delta p$

- Find the optimal $X_s$
- Solve general eigenvalue problem to get $\Delta p$
- Rescale $\Delta p \rightarrow R\Delta p$
- How much along $\Delta p$
  - rescale : Original Linear Optimization Stabilization
  - quartic : line-minimization by fitting a 4$^{\text{th}}$-order polynomial
  - linemin : line-minimization by bracketing
• 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 averages 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)

```bash
$ 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

Compiler options

Use CMAKE_FIND_ROOT_PATH to list XYZ_HOME

Link MKL