Algorithmic and computational challenges for QMC

Paul Kent

Center for Nanophase Materials Sciences
Computer Science & Mathematics Division
Outline

• QMC Background
• Structure of QMCPACK
• Challenges for current & future applications on current & future architectures
  • Running a large enough material system efficiently
  • Development challenges
Acknowledgements
QMCPACK developers
• Jeongnim Kim (Intel)
• Kenneth P. Esler (Stoneridge)
• Miguel Morales (LLNL)
• Anouar Benali (ANL)
• Luke Shulenburger (SNL)
• Jaron Krogel (ORNL)
• Nichols Romero (ANL)
• Raymond Clay (UI)
• ADIOS team (ORNL)
• Many more…

Development currently supported by
• QMC Glue (DOE-BES Predictive Theory and Modeling Program)

Science applications supported by
• DOE-BES and User Facilities
• NSF
• …

Computing resources provided by
• INCITE allocations at Oak Ridge and Argonne Leadership Computing Facilities
• SNL, LLNL
QMCPACK: A production code for science

- Over 400K source lines (C++, templates,...)
- A similar size to major electronic structure packages
- New website http://qmcpack.org
Background

• QMC is – in principle – a systematically improvable electronic structure method applicable to molecules through to solid state systems.

• Energies are (usually) variational.
  – A lower energy indicates a better result
  – Most electronic structure methods are not variational (DFT, many quantum chemical methods)

• QMC can already be applied to current systems of interest where existing methods fail &/or are not predictive.

• Note: In this talk I focus on fixed-node diffusion QMC. Auxiliary Field QMC and full-CI QMC are also attractive but have different strengths/weaknesses.
Obtaining accurate & trustworthy results

1. Simulate a large enough number of atoms (electrons) that the physics/chemistry is well-represented
   - Whole molecule or active site, open boundaries
   - Model region around defect in a material, supercell and periodic boundaries. Twist boundary conditions for metals.

2. Put the atoms in the correct location

3. Use a sufficiently accurate trial wavefunction
   - A good nodal surface minimizes Fermion sign error

If all these points are followed, QMC obtains essentially exact results!
In practice there is a long way to go
Obtaining accurate & trustworthy results

1. Simulate a large enough number of atoms that the physics/chemistry is well-represented
   - Whole molecule or active site, open boundaries
   - Model region around defect in a material, supercell and periodic boundaries. Twist boundary conditions for metals

2. Put the atoms in the correct location

3. Use a sufficiently accurate trial wavefunction
   - A good nodal surface minimizes Fermion sign error

4. Solve the correct Hamiltonian!
   - Use good enough pseudopotentials, if used
   - Eventually need to include relativistic effects
Obtaining accurate & trustworthy results

1. Simulate a large enough number of atoms that the physics/chemistry is well-represented
   - Whole molecule or active site, open boundaries
   - Model region around defect in a material, supercell and periodic boundaries. Twist boundary conditions for metals

2. Put the atoms in the correct location

3. Use a sufficiently accurate trial wavefunction
   - A good nodal surface minimizes Fermion sign error

4. Solve the correct Hamiltonian!
   - Use good enough pseudopotentials, if used
   - Eventually need to include relativistic effects

As we look at more challenging systems with increasingly stringent error demands, all of these areas will require more attention:

Specific science applications will favor specific architectures

Specific architectures will favor certain science applications and improvements in algorithms.

E.g. Balance of processor power/memory size/memory bandwidth
QMC background

Seek the solutions

\[ \hat{H} |\Psi(R)\rangle = E_i |\Psi(R)\rangle \]

\[ \hat{H} = \sum_{i=1}^{N} -\frac{1}{2m_e} \nabla_i^2 + \sum_{i<j} \frac{1}{|r_i - r_j|} + \sum_{i=1}^{N} \sum_{I=1}^{N_{ion}} \frac{-Z_I}{|r_i - r_I|} \]

\[ \langle \Psi_0 | \hat{H} | \Psi_0 \rangle = E_0 \]

Ground-state energy
QMC background

Seek the solutions
\[ \hat{H} |\Psi(R)\rangle = E_i |\Psi(R)\rangle \]

\[ \hat{H} = \sum_{i=1}^{N} -\frac{1}{2m_e} \nabla_i^2 + \sum_{i<j} \frac{1}{|r_i - r_j|} + \sum_{i=1}^{N} v_{ext}(r_i) \]

\[ \langle \Psi_0 | \hat{H} | \Psi_0 \rangle = E_0 \]

Ground-state energy

Variational QMC using Metropolis Sampling

\[ E_T = \frac{\int d^3N R \Psi^*_T(R) \hat{H} \Psi_T(R)}{\int d^3N R |\Psi_T(R)|^2}, \quad E_T > E_0 \]

\[ \Psi_T \]

a trial wavefunction

\[ E_T \approx \frac{\sum_{k=1}^{M} \omega(R_k) \hat{H} \Psi_T(R_k)/\Psi_T(R_k)}{\sum_{k=1}^{M} \omega(R_k)} E_L \]

Need to quickly evaluate (i) ratios of the wavefunction squared, (ii) the local energy
QMC methods

Variational Monte Carlo

- Write down a parameterized form for $\Psi_T$
- Sample distribution $P(R) = |\Psi_T|^2$
  with Metropolis Monte Carlo
- Propose move $r_i \rightarrow r_i'$
- Accept/reject $\propto |\Psi(R')/\Psi_T(R)|^2$
- Average over the distribution
- Minimize $< E_T >$ with respect to the parameters of $\Psi_T$
  (very tricky in practice)

Diffusion Monte Carlo

- Start with VMC optimized $\Psi_T$
- Start with Walkers (population) $R_1, \cdots, R_{N_w^0}, N_w^0 \sim 1000$
  Typically generated by VMC
- Sample distribution
  $P(R) = \Psi_0 \Psi_T$  \hspace{1cm} $\tau = \beta/n$
  $\lim_{\beta \to \infty} e^{-\beta \hat{H}} \Psi_T = \lim_{n \to \infty} [e^{-\tau \hat{H}}]^n \Psi_T$
- Drift/diffuse to move electrons
- Make $M$ copies of each walker
  $M \propto \exp^{-\tau(E_L(R)-E_T)}$
  $E_T = < E_T > - \gamma \log(N_w/N_w^0)$

Very similar computational operations in both algorithms
DMC: computational view

- Light but essential communications
- **Computationally Intensive** : Ratio, Local Energy, & Quantum Force (gradient)

![Diagram](image)

- Population (walkers)
  - Make a move
  - Accept/reject a move
  - Branch with the weight

\[
R' = R + \tau \nabla \ln \Psi_T(R) + \chi
\]

"Quantum Force"

\[
\frac{|\Psi_T(R')|^2}{|\Psi_T(R)|^2} G_d(R \rightarrow R'; \tau)
\]

\[
\exp^{-\tau[(E_L(R) + E_L(R'))/2 - \hat{E}_T]}
\]
**MPI+X Model for QMC**

Each group

```plaintext
for generation = 1 \cdots N_{MC} do
    for walker = 1 \cdots N_w do
        
        end for{walker}
        Reweight and branch walkers
        Update $E_T$ and collect properties
    end for{generation}
```

X on SMP
OpenMP, CUDA, Threads ....
**CPU vs GPU**

```plaintext
for generation = 1 \cdots N_{MC} do
  for walker = 1 \cdots N_w do
    \text{Reweight and branch walkers}
    \text{Update } E_T \text{ and collect properties}
  end for\{walker\}
end for\{generation\}
```

Walkers moved one at a time on each thread.

Reorder loops to vectorize over many walkers. All tasks on GPU (~no transfers).
**Trial wavefunctions**

\[ \Psi_T = e^{J_1 + J_2 + \cdots} \sum_k^M C_k D^\uparrow_k(\phi) D^\downarrow_k(\phi) \]

\[ N = N^\uparrow + N^\downarrow \]

**Correlation (Jastrow)**

\[ J_1 = \sum_i^N \sum_{I}^{N_{ions}} u_1(|r_i - r_I|) \]

\[ J_2 = \sum_{i \neq j}^N u_2(|r_i - r_j|) \]

**Anti-symmetric function (Pauli principle)**

\[ D^\sigma_k = \begin{bmatrix} \phi_1(r_1) & \cdots & \phi_1(r_{N\sigma}) \\ \vdots & \vdots & \vdots \\ \phi_{N\sigma}(r_1) & \cdots & \phi_{N\sigma}(r_{N\sigma}) \end{bmatrix} \]

**Single-particle orbitals**

\[ \phi_i = \sum_l^l c^i_l \Phi_l \]

**Basis sets:** molecular orbitals, plane-wave, grid-based orbitals …
Single-Particle Orbitals

\( \phi(\mathbf{r}_i) \) general function of \((x, y, z)\)

- Evaluate \(N/2\) orbitals at once, \(N \sim 300-3000\)

3D cubic B-spline most efficient for large scale systems

- Strictly local basis set
- Only 64 non-zero elements at \(\mathbf{r}_i\)
- *Fixed cost per-orbital indep. of system size (volume)*

- Memory bandwidth bound
- Uses a lot of memory (GiB) - big problem for “large” systems
- More approximate, less memory costly basis sets available (tradeoffs, no clear win)
Speeding wavefunction evaluation

• Avoid recalculation of wavefunction components
  – Buffer orbitals, Jastrow, on a per electron basis
  – An easy memory vs cost tradeoff

• Store inverse cofactors of determinants, exploit rank-1 update tricks, particularly for multideterminants [major memory, CPU saving]

\[
\frac{D(r_1', r_2, \ldots r_N)}{D(r_1, r_2, \ldots r_N)} = \sum_{j}^{N} \phi_j(r_1')c_{1j}
\]

• We now efficiently evaluate simultaneous row & few column updates. Essential for molecular calculations.
Kernel optimization

- There are many kernels to optimize, most not available in libraries
- We can not take advantage of e.g. quantum chemistry libraries since they usually evaluate integrals, while we need values, gradients, and laplacian
- Substantial human efforts required to optimize: e.g., > 100 CUDA kernels were written by super developer for < 20% of the features of CPU code
- The most important kernels are heavily optimized (10?)
- Hand-tuned (!) spline evaluation routines for Intel/AMD SSEn, IBM QPX, CUDA
- Sadly, it is still possible to beat the compiler by substantial amounts
Simple readable example from multi_bspline_eval_sse2_d_Impl.h

// Main computation loop
for (int i=0; i<4; i++)
    for (int j=0; j<4; j++)
        for (int k=0; k<4; k++)
        {
            __m128d abc, d_abc[3], d2_abc[3];
            abc = _mm_mul_pd(_mm_mul_pd(a[i], b[j]), c[k]);
            d_abc[0] = _mm_mul_pd(_mm_mul_pd(da[i], b[j]), c[k]);
            d_abc[1] = _mm_mul_pd(_mm_mul_pd(a[i], db[j]), c[k]);
            d_abc[2] = _mm_mul_pd(_mm_mul_pd(a[i], b[j]), dc[k]);
            d2_abc[0] = _mm_mul_pd(_mm_mul_pd(d2a[i], b[j]), c[k]);
            d2_abc[1] = _mm_mul_pd(_mm_mul_pd(a[i], d2b[j]), c[k]);
            d2_abc[2] = _mm_mul_pd(_mm_mul_pd(a[i], b[j]), d2c[k]);

            __m128d* restrict coefs = (__m128d*)(spline->coefs +
                (ix+i)*xs + (iy+j)*ys + (iz+k)*zs);

            for (int n=0; n<Nh; n++)
            {
                mvals[n] = _mm_add_pd(mvals[n],
                    _mm_mul_pd(abc, coefs[n]));
                mgrads[3*n+0] = _mm_add_pd(mgrads[3*n+0],
                    _mm_mul_pd(d_abc[0], coefs[n]));
                mgrads[3*n+1] = _mm_add_pd(mgrads[3*n+1],
                    _mm_mul_pd(d_abc[1], coefs[n]));
                mgrads[3*n+2] = _mm_add_pd(mgrads[3*n+2],
                    _mm_mul_pd(d_abc[2], coefs[n]));
                mlapl[3*n+0] = ....
            }
        }
Vampir trace of a 256-el system

A MC step of 256 walkers per GPU, 64 GPUs (MPI tasks)

All_reduce: collect energies and prepare a load balance
p2p to swap walkers (load balance)
Performance

• Achieved a sustained >1 PF performance on the Blue Waters at NCSA (Cray XE6), around 15% of peak
  – Test system: 432 hydrogen atoms under pressure.
  – GPU code ~4x CPU on a per node basis on titan.
  – Efficiency not impressive compared to “GEMM-codes”
  – Parallel efficiency (scaling) is good to 1.5M cores on Sequoia BG

• Why not so “efficient”?
  – Random memory access
  – We have removed dense linear algebra (BLAS) by lower (peak) performing but overall faster algorithms
  – Potentially could do 5-10% better with concerted effort optimizing for specific platforms, but a 2x performance increase would only reduce error bar by $\sqrt{2}$. [ Heresy! ]
  – Method and algorithm development is more important and the payoff could be much greater!
A lesson & a question

- Performance: Need automatic code generation and optimization of key kernels on different platforms
  - Was barely acceptable for a human to generate this code on even one platform
  - It is likely the current code can be improved
  - How to go about this?
Outline

- QMC Background
- Structure of QMCPACK
- Challenges for current & future applications on current & future architectures
  - Running a large enough material system efficiently
  - Development challenges
Single-Particle Orbitals

\( \phi(r_i) \) general function of \((x,y,z)\)

- Evaluate \( N/2 \) orbitals at once
- Need smooth gradient & laplacian
- Used 3D cubic B-splines
- New high accuracy pseudopotentials require very fine grids

Various solutions being (re)developed based on physics

- Hybrid representation
- Mixed grid

Calculations for ZnO and Ca\(_2\)CuO\(_3\) were stopped when we ran out of memory/node, not when we ran out of computer time!
Improving time to solution

- We are looking to divide the work of updating one walker over several threads to reduce time to solution, sacrificing some computational efficiency.
- Cost of moving a walker increases with system size
- Currently only one thread works on any one walker, both CPU & GPU.
- For large systems, we have sufficient electrons to exploit another vector direction (transition?)
- OpenMP tasks? Intel TBB? GPU? Need a long term stable solution for identifying and processing suitably sized chunks of work
Sustainable development

• We need a way to develop sustainably without relying on “super developers”

• We likely have 3 platforms to develop for
  – CPU (OpenMP)
  – GPU (CUDA)
  – Phi (?)

• Need to avoid the rewrite problem – not sustainable or affordable

• Our algorithm is not rich in, e.g. #directivable loops

• Suggestions?
Summary

• Using MPI+X we have developed a high-performing QMC code for SMP and GPU systems
• Memory limitations are a challenge for calculations on large systems
• A transition to a sustainable development model is imperative
QMC Training, 14-18th July 2014