Lessons-learned developing performance portable QMCPACK

IDEAS Seminar May 2023

Paul Kent (kentpr@ornl.gov)
Oak Ridge National Laboratory
Outline

• Brief introduction to Quantum Monte Carlo & QMCPACK
• Performance portability goals
• Challenges of using GPUs
• Development approach
• Summary

Aim: illustrate for other developers & code owners what has been productive for us and our ongoing pain points.
Acknowledgements

ECP QMCPACK team including

• Peter Doak (ORNL)
• William Godoy (ORNL)
• Ye Luo (ANL)

ECP SOLLVE project [OpenMP+LLVM]

OLCF, ALCF staff

AMD, Intel, NVIDIA, HPE engineers
Quantum Monte Carlo

- The most accurate, general approach for solving Schrodinger’s equation for “real” materials. [Foulkes RMP 2001]

- The few approximations in QMC can be tested, unlike standard methods. Nominally $N^3$. **Tradeoff: large computational cost.**

- Not exact, but very accurate today, can treat “strong” electron correlation, applicable to metals, insulators & molecules.

- For details and tutorials, see QMCPACK YouTube channel & https://github.com/QMCPACK/qmc_workshop_2021

\[
\frac{\partial |\psi\rangle}{\partial \tau} = -\hat{H} |\psi\rangle
\]

\[
|\psi(\delta \tau)\rangle = \sum_{i=0}^{\infty} c_i e^{-\epsilon_i \delta \tau} |\phi_i\rangle
\]

Anatase

Rutile

**J. Trail et al. PRB 95 121108 (2017)**

**Y. Luo et al. NJP 18 113049 (2016)**
Performance Portability Goals

1. Run performantly on the full range of hardware, from laptops through to the #1 HPC machine and all 3 main vendor GPUs.

2. Use a single code path on all architectures, to the extent possible. Minimize maintenance burden, increase quality.

3. Retain ability to use specialized hardware & software, where merited.
QMCPACK

- Open source, openly developed on GitHub, ~quarterly releases. Contributors credited on citation papers.
- C++17, HDF5, OpenMP+optional CUDA/HIP/SYCL+vendor dense linear algebra libraries. Highly vectorized, mixed precision supported.
- O(2x10^5) source lines.
- Science production using OpenMP target offload on NV GPUs with release versions of LLVM.
- New design has flexible dispatch, solves data movement and CPU fallback problem. Will always run unlike "legacy" GPU version.
- Code has undergone several major transitions: AoS to SoA CPU code for KNL, removal of "legacy" GPU version, ongoing removal of old CPU code paths.
Recent QMC studies using QMCPACK

Crl₃ monolayers Staros JCP 156 014707 (2022), H phase diagram Niu PRL 130 076102 (2023), >10³ molecules Huang JCTC 19 1712 (2023). We aim to support and accelerate all of these calculations.

O(10³) electrons

O(10²) electrons

O(10¹-2) electrons
Challenge of exploiting GPUs

- Exascale-generation GPUs from NVIDIA, AMD, and Intel have $>10^4$ compute elements. Need $>10^6$ similar operations in flight for optimum performance.
- If we only have $10^{2-4}$ electrons, there naively will not be enough work.
- Generally, no single hot kernel. Kernels are both compute & memory bound.
- Few proven designs. QMC less mature than, e.g., quantum chemistry and classical molecular dynamics where multiple performant implementations are available.
Parallel Scalability

Despite needing communications every timestep, scalability is high due to high computational cost/step, careful MPI implementation.

See Kim et al. JPCM (2018) 10.1088/1361-648X/aab9c3
Key operations

Real space QMC uses both particle-based and dense linear algebra operations. Particle counts + matrix sizes can be small ($10^2$-$10^4$), requiring different choices to standard classical molecular dynamics or quantum chemistry techniques.

Particle operations
- movement, interparticle distances, functions of position, minimum image when periodic

Dense vector, matrix operations
- Spline and Gaussian basis set evaluation, determinant update, wavefunction optimization, BLAS1-3

Example Qs: Is it worth maintaining neighbor lists? Benefits, tradeoffs from sparsity?
Minqmc miniapp for design & performance experiments

- https://github.com/QMCPACK/miniqmc
- Order of magnitude smaller than QMCPACK
- Resulted in new design of QMCPACK with revised internal APIs and flexible runtime dispatch.
- Picked OpenMP target offload as default implementation route, supplemented if needed by vendor specific optimized code.
- Miniapp requires ongoing effort to maintain and update to keep synced with main application. => Unfortunately, miniqmc is currently out of date…

Profile validated vs main app ("proxy not imposter")

Easier profiling of miniapp aids in testing multithreaded offload strategy
Algorithmic Challenge – How to map QMC to GPUs?

- Works well on CPUs. Usually, 1 CPU thread per Monte Carlo walker.

- For GPUs, simply offloading the compute for each walker is not efficient since there is not enough numerical work.
Previous GPU approach:
Batching many independent walker moves

**Batched Metropolis QMC Algorithm, M walkers/node**

do time step i [ 1K-100K ]

    do electron k [ N=10²-10⁴ ]
        propose M new positions \( \{ \mathbf{r}_k' \}_M \) [ \( O(\sim M \cdot MN^2) \) cost ]
        evaluate \( \{ \Psi(\mathbf{r}_k') \}_M \) [ \( O(\sim MN \cdot MN^2) \) cost ]
        accept/reject using \( \sim \left| \Psi' \right|^2 / \left| \Psi \right|^2 \)
        if (accept) update \( \{ \Psi \}_M \) [ \( O(MN^2) \) cost ]
    end k

evaluate Hamiltonian, Observables for all M walkers

spawn/kill walkers, load balance

end i

• Batch (group) all operations over M walkers (Markov chains), now operations are \( O(M \times N) \) or \( O(M \times N^2) \). Choose M large enough to saturate GPU, typically 10-1000.

• CUDA version runs very efficiently, Esler et al. CISE 14 40 (2012)
New approach: multithreaded offload
Batching smaller “crowds” of walkers

- Use multiple smaller batches ("crowds") launched from different host threads, not a single large batch.
  - Trades some kernel efficiency for more asynchronous work and potentially greater throughput. Highly dependent on problem, hw+sw stack.
  - Can recover original GPU algorithm with 1 crowd/thread.
  - Highly beneficial if there is any significant CPU work remaining.

Illustrative only. Not to scale.
Results: 128 atoms NiO / 1536 electrons

Summit@OLCF
Power9+NVIDIA V100
Results: 128 atoms NiO / 1536 electrons
Results: 128 atoms NiO / 1536 electrons

>2x Increase in throughput
Revisiting key algorithms

• We have replaced older algorithms with more compute bound/less memory bound algorithms.

• New delayed update algorithm counterintuitively increases the operation count for higher performance.

• Matrix multiply rich but extra work per step. Uses Sherman-Morrison-Woodbury formula to obtain wavefunction ratios during a delay period $n$, then update inverse. Avoid recalculating intermediates. Improves on our earlier algorithm (McDaniel 2017).

• $\sim$2x faster on GPUs, $\sim$10x faster on CPUs.
Development approach
General approach

- Focus on making the best use of our time
- Pragmatically adopt “best practices”. Refine based on actual data from code review, CI, tools experience.
- Keep barrier for new developers, open-source contributions low.
- Limit required dependencies. Define a support policy for compilers, libraries etc.

E.g. Transitioned documentation to use sphinx & readthedocs. Minimal barrier to doc edits, plus full CI on changes.
Ecosystem challenges: version control of dependencies

• Even “low dependency” apps have many dependencies. These are all undergoing development… changing GPU support, python module changes, HDF5 API updates etc. can all lead to breakage.

• For nightly testing, “version control” achieved via spack package manager (https://spack.io/) to cover a sparse matrix of older/newer software. However, users use ~any combination of versions…
Testing and Continuous Integration

• Large set of unit, deterministic, & stochastic integration tests built up. Test subset run in CI, plus more extensive sets nightly and weekly for many different compiler, CPU, GPU, library combinations.

• Helps us make large changes to the code, onboard new developers, engage with vendors, contributors.

• Pull requests undergo review, testing, coverage reporting, sanitizer tests. Procedures for reviews developed, e.g., merger can not be at same institution as PR.

• CI uses GitHub actions, plus our own hardware, needed to test GPUs not available in cloud. Aim for $O(1h)$ turnaround. Window for input to vendors for fixes in their next release is small.

• OpenMP target offload “GPU” code partly tested via offload to host CPUs with LLVM. Huge time, $ savings.
Many edge case bugs found & resolved

For QMCPACK problems, testing regimen minimizes chance of recurrence. Statistics let us (re)focus testing effort on most critical areas. For external problems, comprehensive testing allows us to give prompt feedback to the relevant developers.

Examples

• QMCPACK:
  – State machines associated with efficient Monte Carlo.
  – Handling of optional features via legacy #ifdefs requiring separate builds.

• Wider software stack:
  – Compilers, particularly OpenMP offload. Problems with complex reductions, multithreading... Latest releases of LLVM in production on NV.
  – Libraries. E.g. Numerically incorrect results from GPU dense linear algebra libraries, threading problem in CPU OpenBLAS (fixed promptly).
  – Many transient packaging and compatibility issues associated with specific versions of libraries, compilers, tools. Important to have latest versions promptly in spack, in addition to older versions.
Status

As of LLVM 15.0 (released 9/2022), performance portable version sufficiently close in performance to limited feature legacy GPU implementation for science production, but with full capabilities available.
Ongoing challenges, open questions

• Further maturation of the ecosystem is necessary. Having Frontier, Aurora, Polaris (etc.) in production will help.

• Helpful to have “stable” and “leading edge” machines available for CI. It is not practical for every app team to run their own CI. Lack of access slows development velocity.

• Automated testing at facilities would help catch issues with vendor provided software, MPI, their unique environment etc.

• Can we obtain ~full performance with only OpenMP using newer/revised versions of the standard, or will some CUDA/HIP/SYCL still be required? What is needed in future C++ standards?
Conclusions

• Performance portable QMC is a challenge!
• Performance portable QMCPACK is in science production.
• Next challenges: taming memory usage, science features.
• Modern development practices, particularly testing, has improved code quality, enabled large changes, and increased our efficiency overall. These practices require dedicated resourcing and staff.

Questions, comments? kentpr@ornl.gov