ALCF COMPUTATIONAL PERFORMANCE WORKSHOP

Argonne NATIONAL LABORATORY

SINGLE-NODE OPTIMIZATION TOWARDS MORE EFFICIENT AND PRODUCTIVE QUANTUM MONTE CARLO SIMULATIONS

QMCPACK

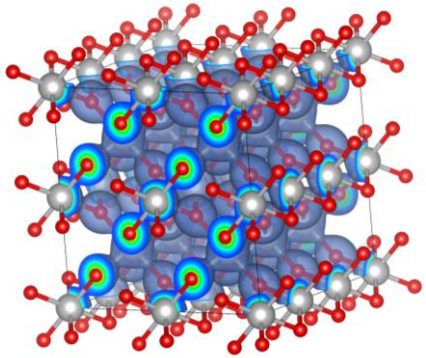
YE LUO
Catalyst Team

Argonne, May 16, 2018

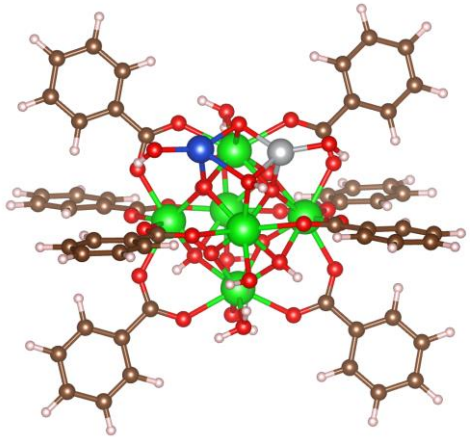
QMCPACK

To answer questions from first principles

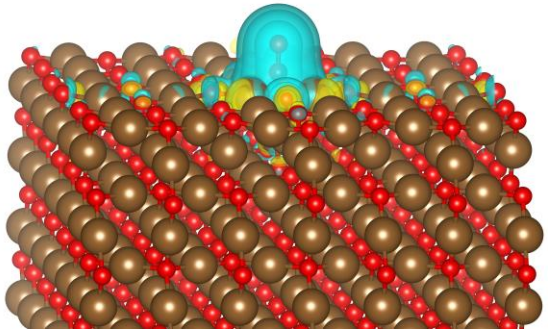
- QMCPACK is an open-source production level many-body ab initio Quantum Monte Carlo code for computing the electronic structure of atoms, molecules, and solids.



Transition metal oxide polymorphs



Metal organic framework



Surface catalysis

QMCPACK TEAM

- Team members spread around the country @ ANL, ORNL, SNL, LLNL

- This work is also contributed by
  - Anouar Benali @ ANL
  - Amrita Mathuriya, Jeongnim Kim @Intel

Supported by:

- ECP AD. QMCPACK: Predictive and Improvable Quantum-mechanics Based Simulations

- Center for Predictive Simulation of Functional Materials, DOE BES

- Intel® Parallel Computing Centers

OUTLINE

- QMC basics
- Optimization is all about memory
  - Where is the bottleneck?
  - Mixed precision
  - New algorithms
  - AoS to SoA
  - How about BGQ?

QUANTUM MONTE CARLO BASICS

QUANTUM MONTE CARLO METHODS

- Many body Schrodinger equation is NP hard

$$\hat{H}\Psi = i\hbar\frac{\partial}{\partial t}\Psi$$

- Quantum Monte Carlo methods allow for a direct treatment and description of complex many-body effects encoded in the wave function, going beyond mean field theory and offering an exact solution of the many-body problem in some circumstances.

VARIATIONAL MONTE CARLO (VMC)

From the variational principle.

- The goal is to solve Schrodinger equation with Monte Carlo technique.

$$\langle E \rangle = \frac{\int \mathrm{d}R \; \Psi^*_{\mathrm{T},\alpha} H \Psi_{\mathrm{T},\alpha}}{\int \mathrm{d}R \; |\Psi_{\mathrm{T},\alpha}|^2} \; ,$$

- Monte Carlo methods can be used to evaluate multi-dimensional integrals much more efficiently than deterministic methods.

$$\rho(R) = \frac{|\Psi_{\mathrm{T},\alpha}(R)|^2}{\int \mathrm{d}R \; |\Psi_{\mathrm{T},\alpha}|^2} \; .$$

- The random walking is performed by many walkers on individual Markov Chains.

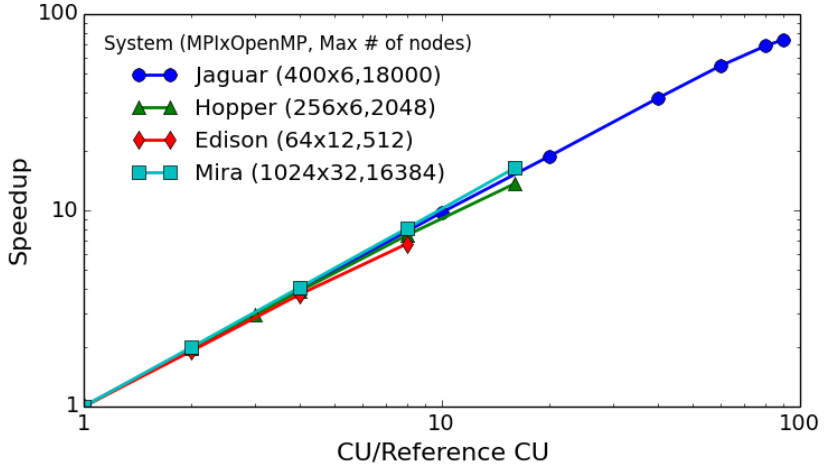
$$E_L(R) = \frac{\hat{H}\Psi_T(R)}{\Psi_T(R)}$$

$$\langle E \rangle = \sum_R E_L(R)$$

DMC ALGORITHM

Multiple levels of parallelism can be exploited.

1: **for** MC generation $= 1 \cdots M$ **do**
2: **for** walker $= 1 \cdots N_w$ **do**
3: let $\mathbf{R} = \{\mathbf{r}_1 \ldots \mathbf{r}_N\}$
4: **for** particle $i = 1 \cdots N$ **do**
5: set $\mathbf{r}_i^{'} = \mathbf{r}_i + \delta$
6: let $\mathbf{R}^{'} = \{\mathbf{r}_1 \ldots \mathbf{r}_i^{'} \ldots \mathbf{r}_N\}$
7: **ratio** $\rho = \Psi_T(\mathbf{R}^{'})/\Psi_T(\mathbf{R})$
8: **derivatives** $\nabla_i \Psi_T, \nabla_i^2 \Psi_T$
9: **if** $\mathbf{r} \rightarrow \mathbf{r}'$ is accepted **then**
10: update state of a walker
11: **end if**
12: **end for**{particle}
13: **local energy** $E_L = \hat{H}\Psi_T(\mathbf{R})/\Psi_T(\mathbf{R})$
14: reweight and branch walkers
15: **end for**{walker}
16: update E_T and load balance
17: **end for**{MC generation}



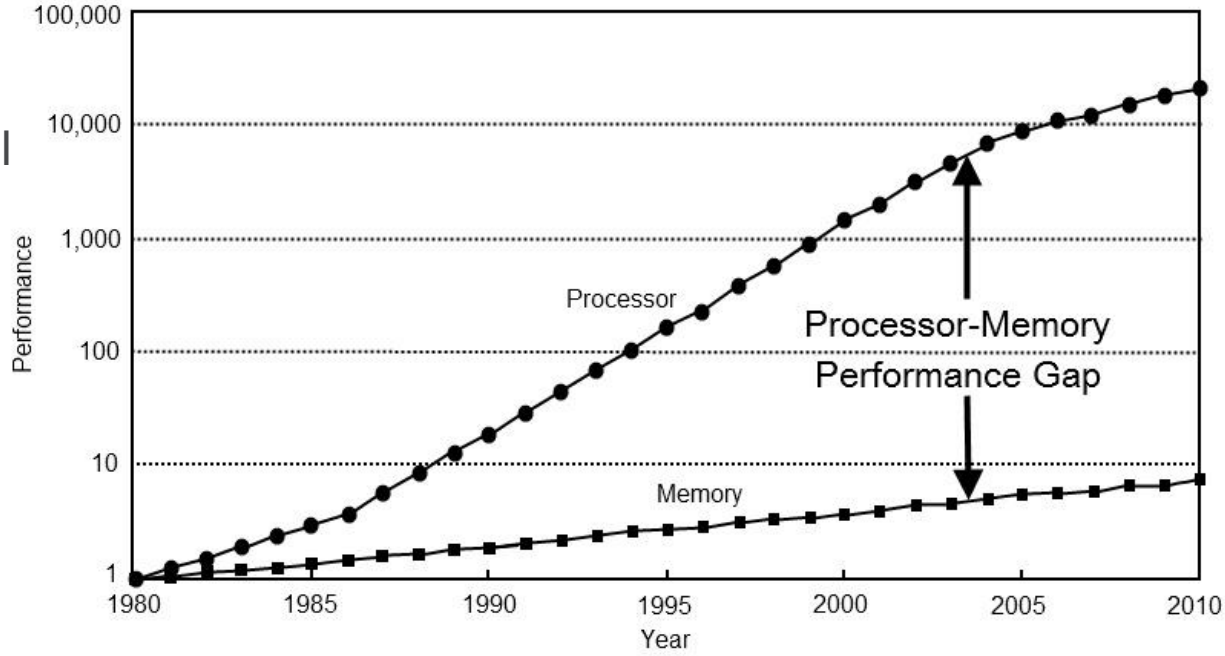
- Loop 2: walkers are distributed both over MPI and cores/SMs.
- Loop 2 and 4 are interchanged on GPU.
- Steps 6,7,8 have extra particle concurrency, exposed to GPU currently but extensible to SIMD.

OPTIMIZATION IS ALL ABOUT MEMORY

PROCESSOR-MEMORY PERFORMANCE GAP

Getting worse

- KNL 7230:
  DDR4-2400 6 channel
  115.2 GB/s
  MCDRAM 400+ GB/s
- GTX 1080 Ti
  GDDR5X 484 GB/s
- Tesla P100
  HBM2: 732 GB/s



Computer Architecture: A Quantitative Approach by John L. Hennessy, David A. Patterson, Andrea C. Arpaci-Dusseau

SPECIFICATION OF KNL AND BLUEGENE/Q

A big step from multi-core to many-core era

	KNL (Phi 7210)	BG/Q	Ratio
Core counts	64	16	4
Hardware threads	4	4	1
Core design	Out-of-order	In-order	
FPU	2 x 512 bit	256 bit	4
Vector ISA	AVX512	QPX	
FLOPs/Cycle	8/16 way DP/SP	4 way DP/SP	2/4 DP/SP
Base clock	1.3 GHz	1.6 GHz	0.813
DP Flops	2.662 T	0.205 T	13
High Bandwidth Mem.	16GB (>400GB/s)	0	9.4 to DDR
Memory	192 GB (102 GB/s)	16 GB (42.6 GB/s)	12 (2.4)

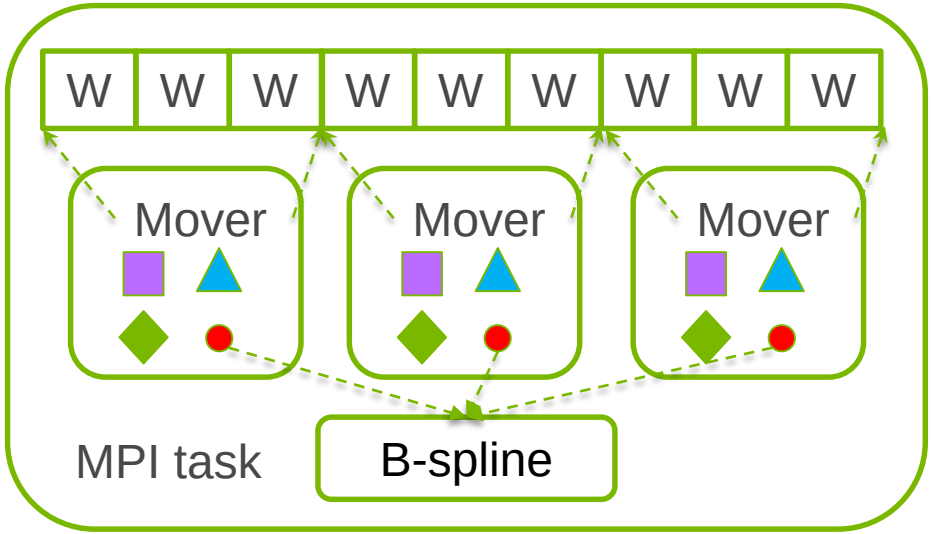
FEED DATA FASTER

Use all the possible methods

- Use single or even half precision?
  - BGQ less BW, GPU/KNL more throughout
- Reduce memory footprint to fit fast memory
  - Better performance for cache miss. DDR vs MCDRAM, GDDR vs PCI-e
- Improve algorithms following computer-over-store policy
  - Less BW demand
  - Improve data locality for better cache efficiency
- Use vector friendly operations
  - Maximize cache and vector unit performance

SINGLE-NODE MANAGEMENT

Fat walker creates problem



- Each mover sits on a threads
- A few walkers are propagated by a mover.
- Data staged from walker to mover and copied back after propagation.
- 1 walker per thread is the lower limit for time to solution.
- Fat walker problem occurs.

MEMORY USAGE

Components and scaling in the TiO2 supercell calculation

- Shared by all the threads
  - Single particle orbitals (B-spline coefficients)
  - Const. data: Ionic distance table
- Walker owned buffer
  - Slater determinants (SD)
  - Two body potential matrix elements (Jastrow)
  - Particle coordinates
- Thread owned work space
  - Staged SD and Jastrow, Hamiltonian
  - Ion-electron, electron-electron distance tables
  - Extra scratch space for computing

Data	Scaling	Size
Shared	N	2.5 GB
Thread	$T*N^2$	7.6 GB
Walker buffer	$W*N^2$	5.0 GB
Total		15.1 GB

single/double precision

- Thread scratch space is reduced by directly using walker owned buffers.

OPTIMIZATION PART I, MIXED PRECISION

DOUBLE TO SINGLE PRECISION

More computing power and less memory pressure

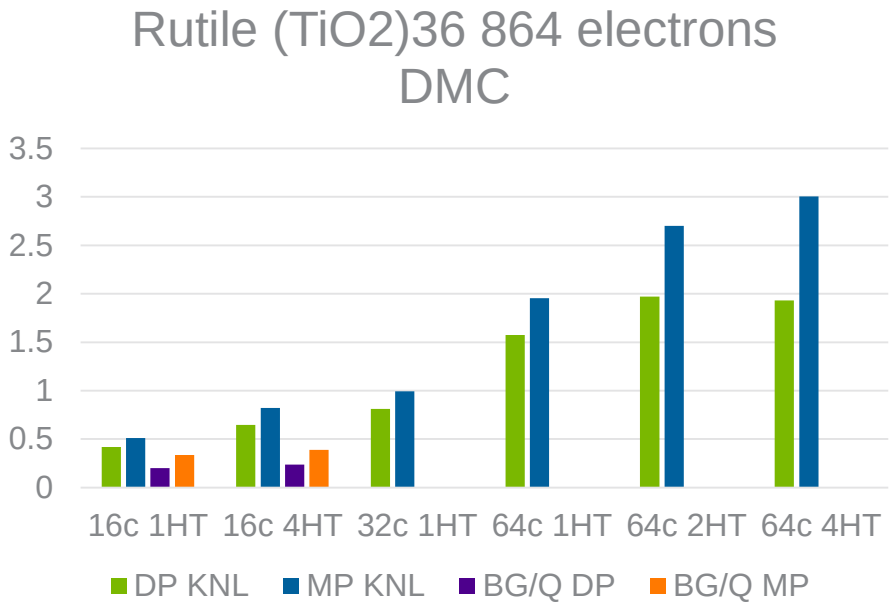
- Hardware capability DP/SP flops.
  - Intel Xeon or Xeon Phi 1/2
  - Nvidia Kepler 1/3, P100 1/2
  - IBM BlueGene/Q 1/1
- Reduce memory bandwidth demand.
- Reduce memory footprint, fit into HBM.

- In QMCPACK, GPU accelerated parts are mostly in SP except matrix inversions and coulomb interaction. Host side is always in DP.
- SP achieves 2X DP performance on K80.

WEAK SCALING PERFORMANCE

Gain performance not only on KNL but also on BG/Q.

- Small core counts: 20% faster on KNL, gained from computing.
- Full node: 55% faster, gained from both computing and memory BW.
- Always: about 70% faster on BG/Q, gained from memory BW.



MEMORY USAGE

Components and scaling in the TiO2 supercell calculation

- Shared by all the threads
  - Single particle orbitals (B-spline coefficients)
  - Const. data: Ionic distance table
- Walker owned buffer
  - Slater determinants (SD)
  - 2-B potential matrix elements (Jastrow)
  - Particle coordinates
- Thread owned work space
  - Staged SD and Jastrow, Hamiltonian
  - Ion-electron, electron-electron distance tables
  - Extra scratch space for computing

Data	Scaling	Size
Shared	N	2.5 GB
Thread	$T*N^2$	7.6->3.8 GB
Walker buffer	$W*N^2$	5.0->2.5 GB
Total		15.1->8.8 GB

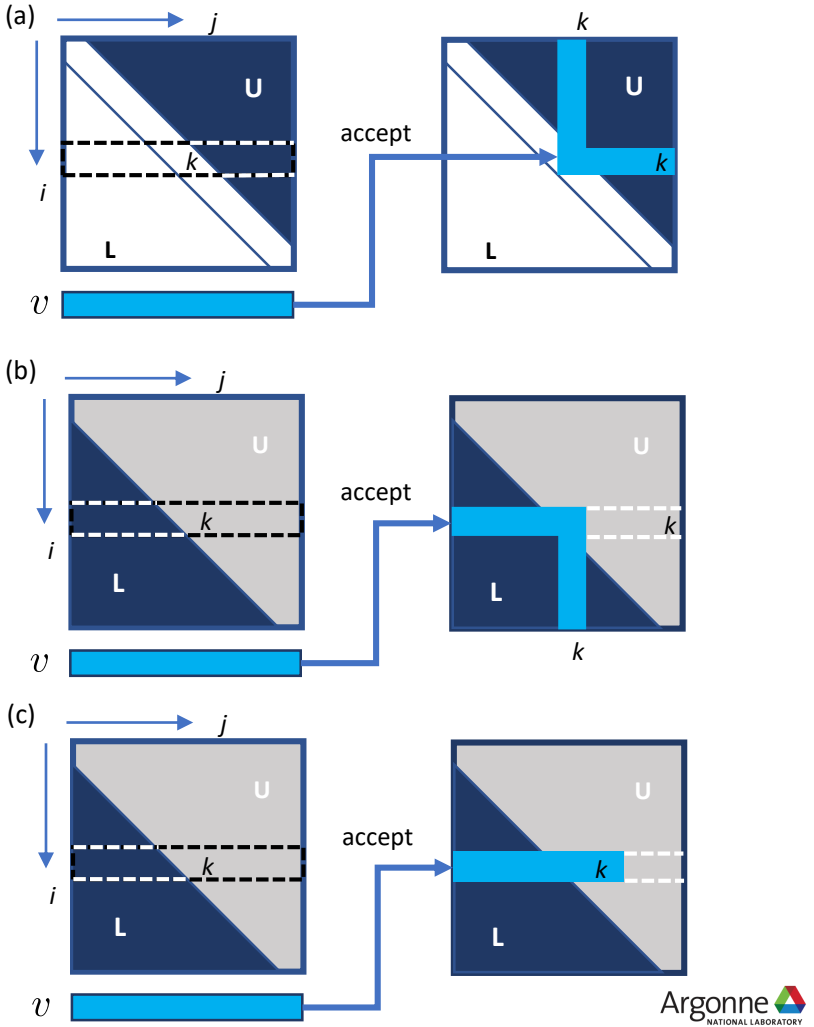
single/double precision

OPTIMIZATION PART II, ALGORITHMS

DISTANCE TABLES

Faster memory access

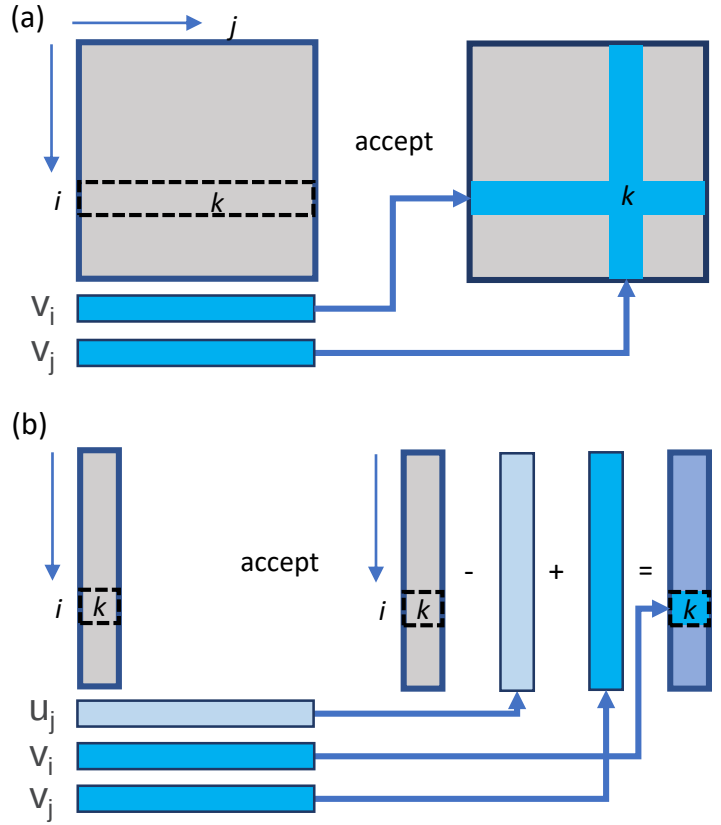
- Consumed by wavefunction (Jastrow), Hamiltonian (Coulomb potential) and properties (structure factor)
- (a) Compact ½ storage but not aligned access for both row and column updates.
- (b) Use full storage, but always aligned access for both row and column updates. Row coalesced, column striped.
- (c) Only row update. Forward computing. Remove upper triangle.



TWO/THREE BODY JASTROW FACTOR

Use an update scheme

- Two/Three body potential U(i,j)
- U(i,j)=U(j,i) and only need row summation.
- Memory footprint N^2=>N
- Reduce the cost prefactor
- All vectorizable computation



FAT TO LEAN WALKER

Deep dive into the walker memory usage

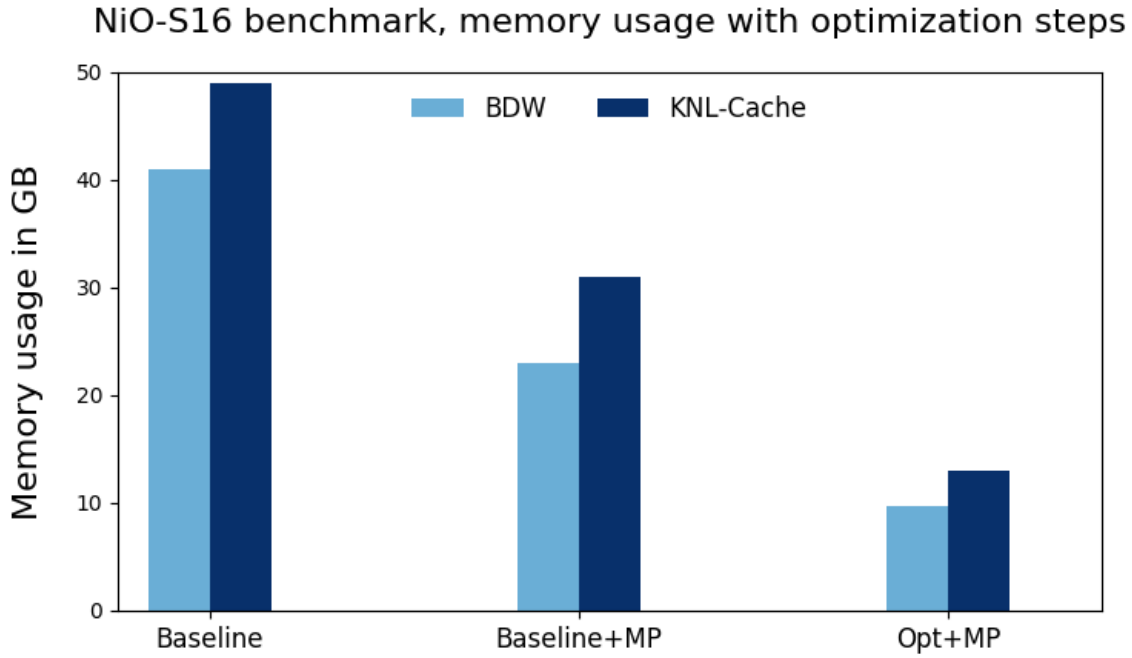
Data	Contents	Size per walker	Size per thread	Total size
Dist. table e-e	Rij, **R**ij, ~~R^{-1}~~	0	$2N^2*5 => 4N^2*4$	$10N^2 => 16N^2$
Slater Det	M, ∇M, ΔM, M_temp…	$5N^2*2$	$11N^2*2 => 5N^2*2$	$32N^2 => 20N^2$
2B Jas	~~Uij, ∇Uij, ΔUij, PairID~~	$5*(2N)^2 => 0$	$6*(2N)^2 => 0$	$44N^2 => 0$
Total		$30N^2 => 10N^2$	$56N^2 => 26N^2$	$86N^2 => 36N^2$

Red contents are identical for the walker and the threads.

Mixed precision needs extra memory($4N^2$) for SD for recomputing in DP.

NIO 64 ATOM BENCHMARK

A huge save in memory footprint, 49GB=>13GB



OPTIMIZATION PART III, AOS TO SOA

AoS => SoA

Data layout change to facilitate vectorization

- For physics, R[N][3], Vector<TinyVector<T,3> >
- For computing, Rsoa[3][N], VectorSoaContainer<T,3>
- Cache aligned and padded inside class.
- Rsoa.data(d) to access d-th array.
- Friendly to #pragma omp simd aligned(x)
- Operator [ ] returns TinyVector<T,3>

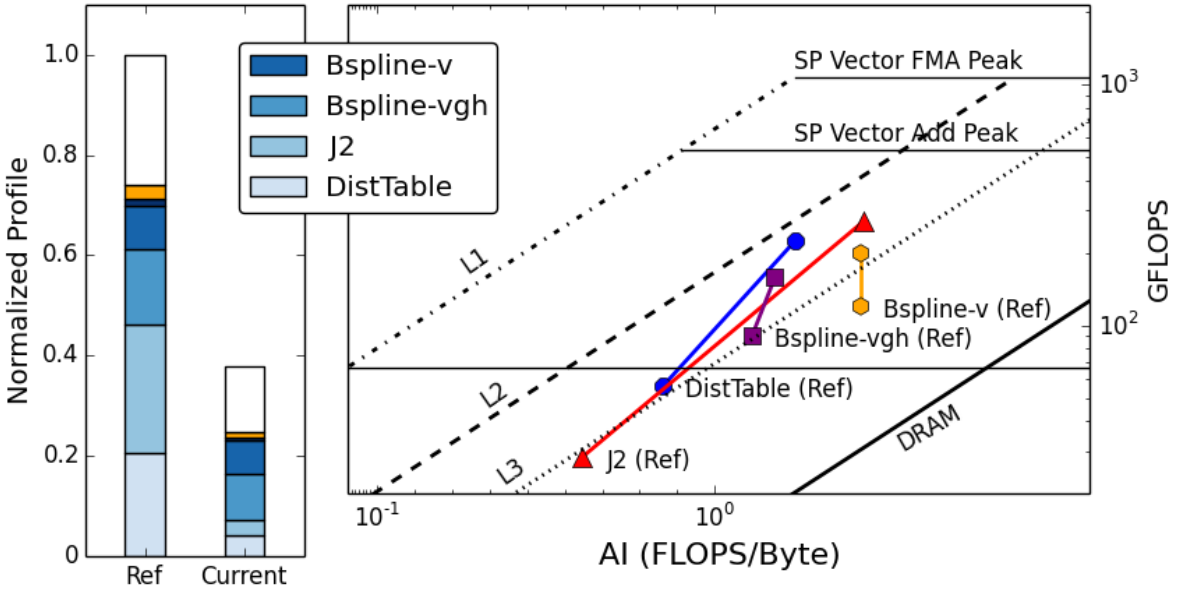
Gradients $\sum_j \nabla U(i,j)$

```
posT grad;
Before:
for(int jat=0; jat<N; ++jat) grad+=du[jat]*displ[jat];
After:
for(int idim=0; idim<DIM; ++idim)
{
  const valT* restrict dX=displ.data(idim);
  valT s=valT();
  #pragma omp simd reduction(+:s) aligned(du,dX)
  for(int jat=0; jat<N; ++jat) s+=du[jat]*dX[jat];
  grad[idim]=s;
}
```

ROOFLINE ANALYSIS

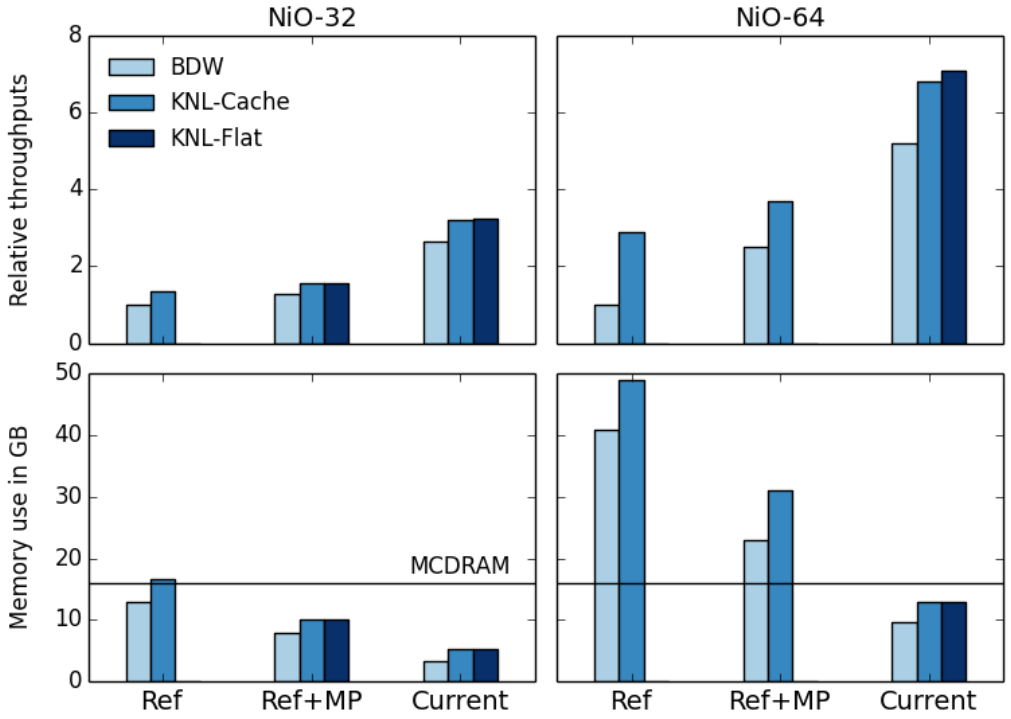
Significant gain in the distance table and the Jastrow

- On BDW, only DDR
- Drop from 47% to 8%
- Huge improvement in algorithmic intensity
- Everything in L3



BENCHMARK

New code saves memory and speeds up the calculation



OPTIMIZATION PART IV, BLUE GENE/Q

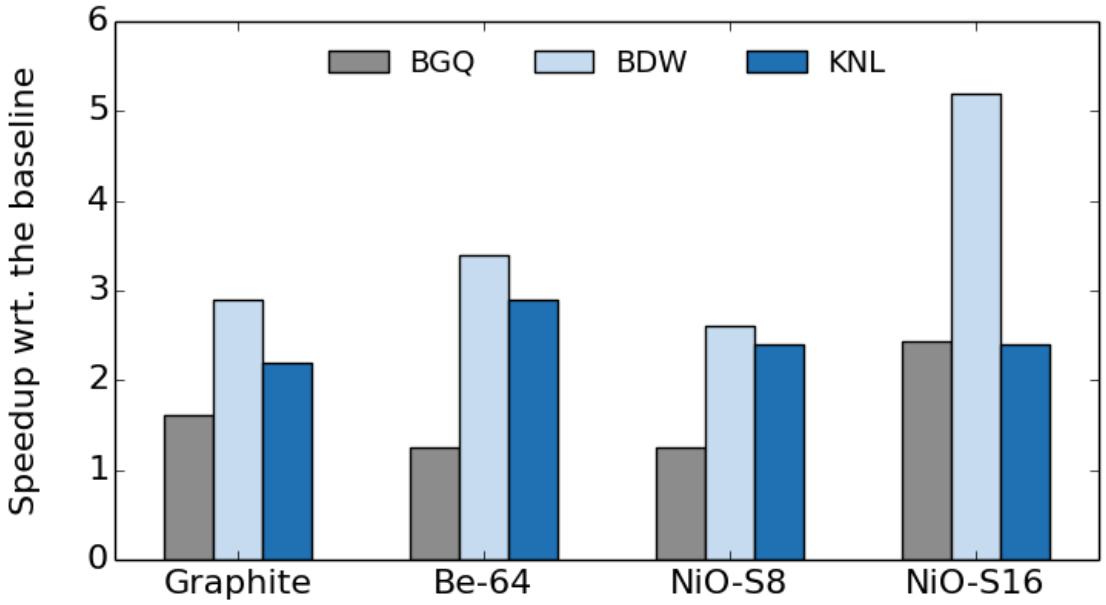
BGCLANG SAVES BG/Q

Support new language features

- BGClang 4.0
- Pros:
  - Supports C++11 with newer STL.
  - Supports OpenMP >=4.0 for #pragma omp simd
  - Supports QPX intrinsics
  - Build codes much faster than XL
  - More warnings
- Cons:
  - Auto-vectorization is bad with single precision but OK with #pragma omp simd
  - Needs effort if esslsmp is needed with OpenMP

BENCHMARK

Significant speed on all the platforms



SUMMARY

Argonne NATIONAL LABORATORY

ACHIEVEMENT

>2X speed up and huge memory saving

- Introduce mixed precision to the CPU code globally. GPU code only has it locally.
- We introduced new algorithms to save memory and speed up the calculation. This is long term benefit and beneficial to all the platforms.
- We refactor the code to support SoA and get ready for future improvement.
- We developed miniQMC including a set of QMCPACK kernels to facilitate dev.

SUGGESTIONS

- Performance tools are your best friend. Probably need more than one for different aspects.
- Working harder on algorithms than implementations
- Data movement is always much more expensive than computing
- Let the compilers do more works
- Work more on miniapps rather than the whole app. The developers' time is more precious.

LET'S CRUSHING MACHINES WITH QMCPACK