Argonne NATIONAL LABORATORY

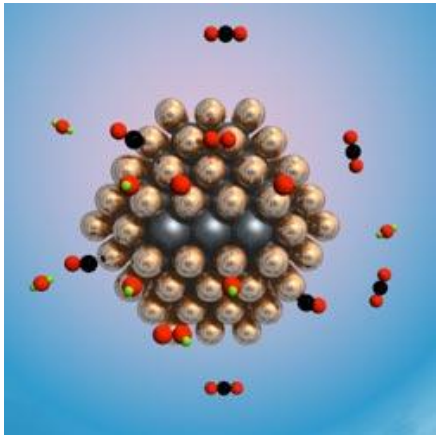
Quantum Monte Carlo on Blue Gene/Q using QMCPACK: QPX application to performance

Anouar Benali

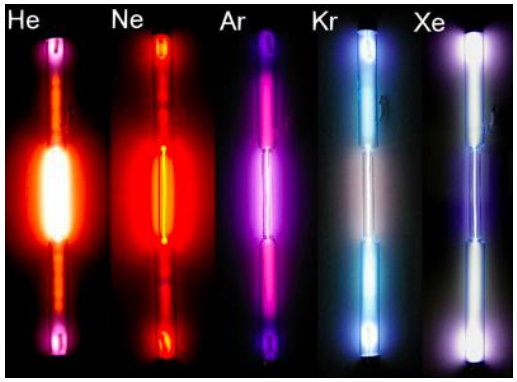
Assistant Computational Scientist

Objectives

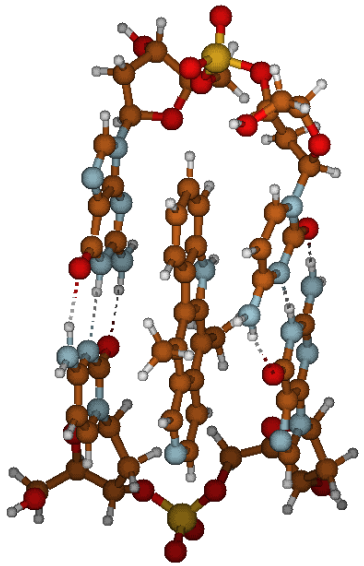
Use Ab initio methods to investigate and understand key properties of energy-related materials for catalysis



Solids, surfaces and Nanoclusters for Catalysis purposes.



Van der Waals dominated systems



Chemistry/Biology

QMCPACK Simulation Suite

- Jeongnim Kim.	Intel (Formerly ORNL)
- David Ceperley	UIUC
- Paul Kent	ORNL
- Luke Shulenburger	SNL
- Ken Esler	Stoneridge (Formerly UIUC)
- Miguel Morales	LLNL
- Jeremy McMinis	Wizeline (Formerly LLNL)
- Anouar Benali	ANL
- Jaron Krogel	ORNL
- Ye Luo	ANL
- Raymond Clay	LLNL/UIUC
- Norm Tubman	UIUC

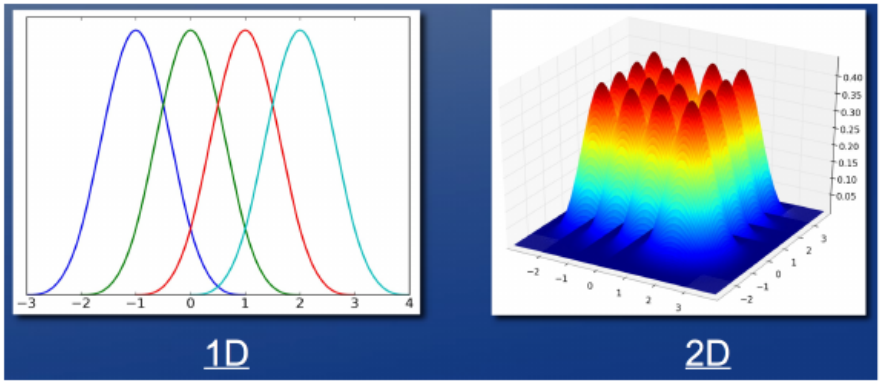
[1] J. Kim *et al.* J. of Phy. - Conf. Series. (2012) vol. 402, no. 1, 012008
[2] K. P. Esler *et al.* Comp. in Sci. and Eng. (2012) vol. 14, no. 1, 40.
[3] J. Kim et al. "Qmcpack simulation suite."

Single particle orbitals

- The most common source for single particle orbitals for solid state calculations is from a DFT code
- These wavefunctions are typically represented in a plane wave basis: $$\phi_i(\vec{r}) = \sum_j c_{ij} e^{\vec{k}_j \cdot \vec{r}}$$
- Each single particle move requires evaluating every orbital at a new position
  - In a plane wave basis set, this means all N plane waves must be evaluated for every orbital!

Single particle orbitals

- Plane wave representations are inefficient
- A real space basis is much more efficient
  - Extensive testing shows that 3D b-splines are a good choice for efficiency
  - Only 64 basis elements are nonzero at each point (versus thousands or more for plane waves)
  - $B_{ijk}(x,y,z) = a_i(x)b_j(y)c_k(z)$



- Efficient routines exist to convert from a plane wave basis to b-splines with an arbitrary real space spacing using the FFT

Performance optimization using QPX intrinsics

Profiling

System:

- Ar Solid – 32 atoms – 256 electrons – B-splines representation of WF (1.9Gb) :
- 256 nodes – 32 threads – 2 Walkers per thread

Profile with original version of QMCPACK

Flat profile:

Total run time: 53min40

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
56.95	58369.57	58369.57				**.eval_multi_UBspline_3d_z_vgh**
14.02	72738.82	14369.25				**.eval_multi_UBspline_3d_z**
2.11	77918.51	2161.01				SymmetricDTD
1.70	79663.07	1744.56				EinsplineSetExtended::evaluate

Evaluation of spline, gradient and hessian coefficients (complex)

Evaluation of spline coefficients (complex)

71% of the application time spent in the Spline evaluation of the Wave Function

Einspline

Eval_z_vgh:

Evaluation of spline coefficients, gradient and hessian (complex)

```
for (int i=0; i<4; i++)
    for (int j=0; j<4; j++)
      for (int k=0; k<4; k++) {
   //missing code (definition of abc
and loading arithmetic pointer coefs)
    for (int n=0; num_splines; n++) {
    vals[n]      +=   abc   *coefs[n];
    grads[3*n+0] +=  dabc[0]*coefs[n];
    grads[3*n+1] +=  dabc[1]*coefs[n];
    grads[3*n+2] +=  dabc[2]*coefs[n];
    hess [9*n+0] += d2abc[0]*coefs[n];
    hess [9*n+1] += d2abc[1]*coefs[n];
    hess [9*n+2] += d2abc[2]*coefs[n];
    hess [9*n+4] += d2abc[3]*coefs[n];
    hess [9*n+5] += d2abc[4]*coefs[n];
    hess [9*n+8] += d2abc[5]*coefs[n];
}
}
```

Strategy

- Inner loop to Outer loop
- Unroll (j) and (k) loops
- Reformulate the arithmetic problem
- Load data from non contiguous

Einspline

```
 complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
    for (int n=0; n<num_splines; n++, coefs++) {
       for (int i=0; i<4; i++) {

          double pre0 =   a[i] *   b[0];
          double pre1 =  da[i] *   b[0];
          double pre2 = d2a[i] *   b[0];
          double pre3 =   a[i] *  db[0];
  complex_double coef0 = coefs[i*xs];
  complex_double coef1 = coefs[i*xs + zs];
  complex_double coef2 = coefs[i*xs + 2*zs];
  complex_double coef3 = coefs[i*xs + 3*zs];
  complex_double sum0 = coef0 * c[0] + c[1] * coef1 + c[2] * coef2 + c[3] * coef3;
  complex_double sum1 = coef0 * c[0] + dc[1] * coef1 + dc[2] * coef2 + dc[3] *
coef3;

      //Code omitted
        // val   +=   pre0 * sum0 + pre01 * sum01 + pre02 * sum02 + pre03 * sum03;
        // grad0  +=
         ..
        }
      vals[n] = val;
      grads[3*n+0] = grad0;
       //code omitted
  }
```


Einspline

Eval_z_vgh:

Evaluation of spline coefficients,

Gradient and Hessian (complex)

Number of Cycles = 1879207

```
  503.975   All XU Instruction
  619.513   All AXU Instruction
1.141.770   FP Operations Group 1
Derived metrics for code block
 "Original" averaged over process(es)
 on node <0,0,0,0,0>:
Instruction mix:  FPU = 55.14 %,
FXU = 44.86 %
Instructions per cycle completed
 per core = 0.5940
Per cent of max issue rate per core = 59.40 %
Total weighted GFlops for this node = 0.966
Loads that hit in L1 d-cache =  93.44 %
                  L1P buffer =   5.52 %
                  L2 cache   =   0.83 %
                  DDR        =   0.21 %
DDR traffic for the node: ld = 0.009,
 st = 0.016, total = 0.025 (Bytes/cycle)
```

Number Of Cycles = 1184355

```
240.030   All XU Instruction
299.508   All AXU Instruction
634.154   FP Operations Group 1
Derived metrics for code block
"Algo B" averaged over process(es)
on node <0,0,0,0,0>:
Instruction mix:  FPU = 55.51 %,
  FXU = 44.49 %
Instructions per cycle completed
 per core = 0.4466
Per cent of max issue rate per core = 44.66 %
Total weighted GFlops for this node = 0.840
Loads that hit in L1 d-cache =  93.68 %
                  L1P buffer =   0.26 %
                  L2 cache   =   5.06 %
                  DDR        =   1.00 %
DDR traffic for the node: ld = 0.024, st =
0.037, total = 0.060 (Bytes/cycle)
```

Speedup of 1.59

Einspline

```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
   for (int n=0; n<num_splines; n++, coefs++) {
      for (int i=0; i<4; i++) {


  complex_double coef00 = coefs[i*xs];
  complex_double coef11 = coefs[i*xs + zs];
  complex_double coef22 = coefs[i*xs + 2*zs];
  complex_double coef33 = coefs[i*xs + 3*zs];


  val = a[i]{(coef00*c[0]+coef01*c[1]+coef02*c[2]+coef03*c[3])*b[0]
            +(coef10*c[0]+coef11*c[1]+coef12*c[2]+coef13*c[3])*b[1]
            +(coef20*c[0]+coef21*c[1]+coef22*c[2]+coef23*c[3])*b[2]
            +(coef30*c[0]+coef31*c[1]+coef32*c[2]+coef33*c[3])*b[3]
            }
```

Reminder:

vector4double : {doubles,double,double,double}

Complex_double : {double,double}

Einspline

```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
   for (int n=0; n<num_splines; n++, coefs++) {
      for (int i=0; i<4; i++) {


val = a[i]{(coef00*c[0]+coef01*c[1]+coef02*c[2]+coef03*c[3])*b[0]

val = a[i]{
   (real(coef00)*c[0]+real(coef01)*c[1]+real(coef02)*c[2]+real(coef03)*c[3]
    imag(coef00)*c[0]+imag(coef01)*c[1]+imag(coef02)*c[2]+imag(coef03)*c[3])*b[0]
    //code omitted
            }



         val = a[i]{
            (real(coef00)*c[0] +  real(coef02)*c[2]
             imag(coef00)*c[0] +  imag(coef02)*c[2]
             real(coef01)*c[1] +  real(coef03)*c[3]
             imag(coef01)*c[1] +  imag(coef03)*c[3]) *b[0]
```


Einspline

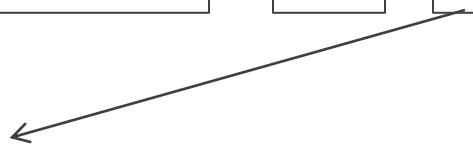
```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
   for (int n=0; n<num_splines; n++, coefs++) {
      for (int i=0; i<4; i++) {


val = a[i]{(coef00*c[0]+coef01*c[1]+coef02*c[2]+coef03*c[3])*b[0]

val = a[i]{
   (real(coef00)*c[0]+real(coef01)*c[1]+real(coef02)*c[2]+real(coef03)*c[3]
    imag(coef00)*c[0]+imag(coef01)*c[1]+imag(coef02)*c[2]+imag(coef03)*c[3])*b[0]
    //code omitted
            }
```

```
        val = a[i]{

  real(coef00) * c[0]     real(coef02) * c[2] * b[0]
  imag(coef00) * c[0]     imag(coef02) * c[2] * b[0]
  real(coef01) * c[1] +   real(coef03) * c[3] * b[0]
  imag(coef01) * c[1]     imag(coef03) * c[3] * b[0]
```



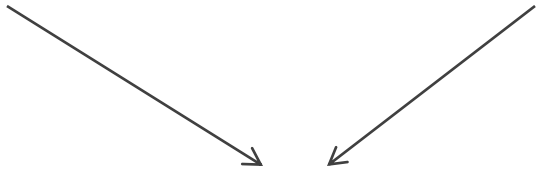
```
vector4double b_0=vec_splat(b,0);
```


Einspline

```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
   for (int n=0; n<num_splines; n++, coefs++) {
      for (int i=0; i<4; i++) {

          val = a[i]{
```

real(coef00)	*	c[0]		**real**(coef02)	*	c[2]	*	b[0]
imag(coef00)	*	c[0]		**imag**(coef02)	*	c[2]	*	b[0]
real(coef01)	*	c[1]	+	**real**(coef03)	*	c[3]	*	b[0]
imag(coef01)	*	c[1]		**imag**(coef03)	*	c[3]	*	b[0]



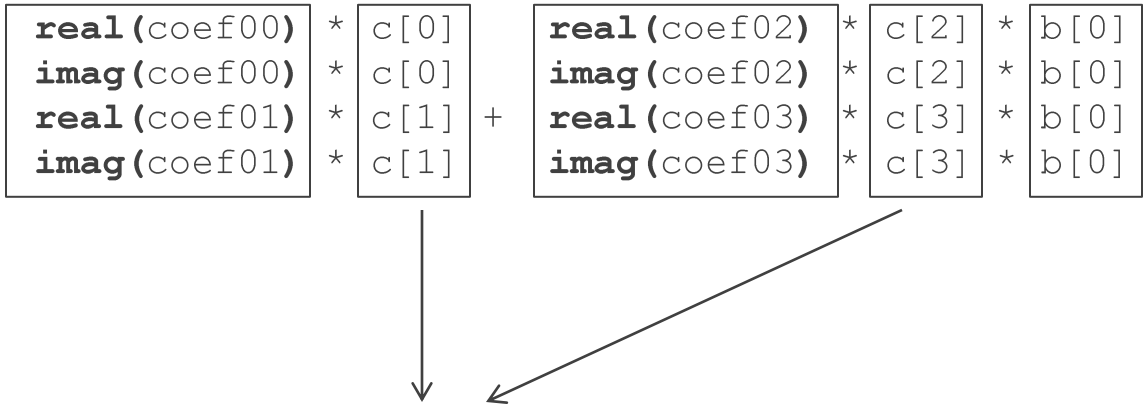
```
vector4double Coef0_0_0=vec_ld2(0,(double*)(coefs+(i*xs)));
vector4double Coef0_0_1=vec_ld2(0,(double*)(coefs+(i*xs+zs)));


  vector4double Coef0_1_0=vec_sldw(Coef0_0_0,Coef0_0_1,2);
```


Einspline

```
complex_double* restrict coefs = spline->coefs + ix*xs + iy*ys + iz*zs;
   for (int n=0; n<num_splines; n++, coefs++) {
      for (int i=0; i<4; i++) {

          val = a[i]{
```




```
    vector4double mantissa_1 = vec_gpci(0011);
    vector4double mantissa_2 = vec_gpci(02233);


      vector4double splatc_0_1=vec_perm(c,c,mantissa_1);
      vector4double splatc_0_2=vec_perm(c,c,mantissa_2);


vector4double Coef01=vec_add(vec_mul(Coef0_1_0,splatc_0_1),vec_mul(Coef0_2_0,splatc_0_2));
```


Einspline

Evaluation of spline coefficients,
Gradient and Hessian (complex)

Number of Cycles = 1879207

```
  503.975    All XU Instruction
  619.513    All AXU Instruction
1.141.770    FP Operations Group 1
Derived metrics for code block
 "Original" averaged over process(es)
 on node <0,0,0,0,0>:
Instruction mix:  FPU = 55.14 %,
FXU = 44.86 %
Instructions per cycle completed
 per core = 0.5940
Per cent of max issue rate per core = 59.40 %
Total weighted GFlops for this node = 0.966
Loads that hit in L1 d-cache =  93.44 %
                 L1P buffer =   5.52 %
                 L2 cache   =   0.83 %
                 DDR        =   0.21 %
DDR traffic for the node: ld = 0.009,
 st = 0.016, total = 0.025 (Bytes/cycle)
```

Number Of Cycles = 246752

```
 52.395    All XU Instruction
 32.802    All AXU Instruction
160.874    FP Operations Group 1
Derived metrics for code block
"Benali QPX" averaged over process(es)
 on node <0,0,0,0,0>:
Instruction mix:  FPU = 38.50 %,
 FXU = 61.50 %
Instructions per cycle completed
per core = 0.3115
Per cent of max issue rate per core = 31.15 %
Total weighted GFlops for this node = 0.941
Loads that hit in L1 d-cache =  75.10 %
                 L1P buffer =   1.00 %
                 L2 cache   =  20.06 %
                 DDR        =   3.84 %
DDR traffic for the node: ld = 0.068, st =
0.121, total = 0.190 (Bytes/cycle)
```

Speedup compared to original: 7.62

Profiling

System:

- Ar Solid – 32 atoms – 256 electrons – Bsplines WF (1.9Gb) :
- 256 nodes – 32 threads – 2 Walkers per thread

Profile with QPX and Prefetch

Flat profile:
Total run time: 20min03
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	
14.08	5380.43	5380.43	**.eval_multi_UBspline_3d_z_vgh**
8.25	12270.83	3152.52	**.eval_multi_UBspline_3d_z**
5.68	14441.45	2170.62	.SymmetricDTD
4.85	16292.97	1851.52	EinsplineSetExtended::evaluate

Profile with Original Algoritm

Flat profile:
Total run time: 53min40
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	
56.95	58369.57	58369.5	
14.02	72738.82	14369.25	
2.11	77918.51	2161.01	
1.70	79663.07	1744.5	

Total run time Speedup of 2.68 times

HPM PROFILING

Original Code

```
27.644.290.379.027   All XU Instruction
22.786.190.220.714   All AXU Instruction
43.043.218.198.088   FP Operations Group 1

Derived metrics for code block "mpiAll"
averaged over process(es) on node <0,0,0,0,0>:
Instruction mix:  FPU = 45.18 %,  FXU = 54.82 %
Instructions per cycle completed per core =
0.6138
Per cent of max issue rate per core = 33.65 %
Total weighted GFlops for this node = 13.412
Loads that hit in L1 d-cache =  94.03 %
                 L1P buffer =   5.36 %
                 L2 cache   =   0.35 %
                 DDR        =   0.26 %
DDR traffic for the node: ld = 1.508, st =
0.540, total = 2.049 (Bytes/cycle)
```

Percentage of peak= 6.55%

BGQ optimized Code

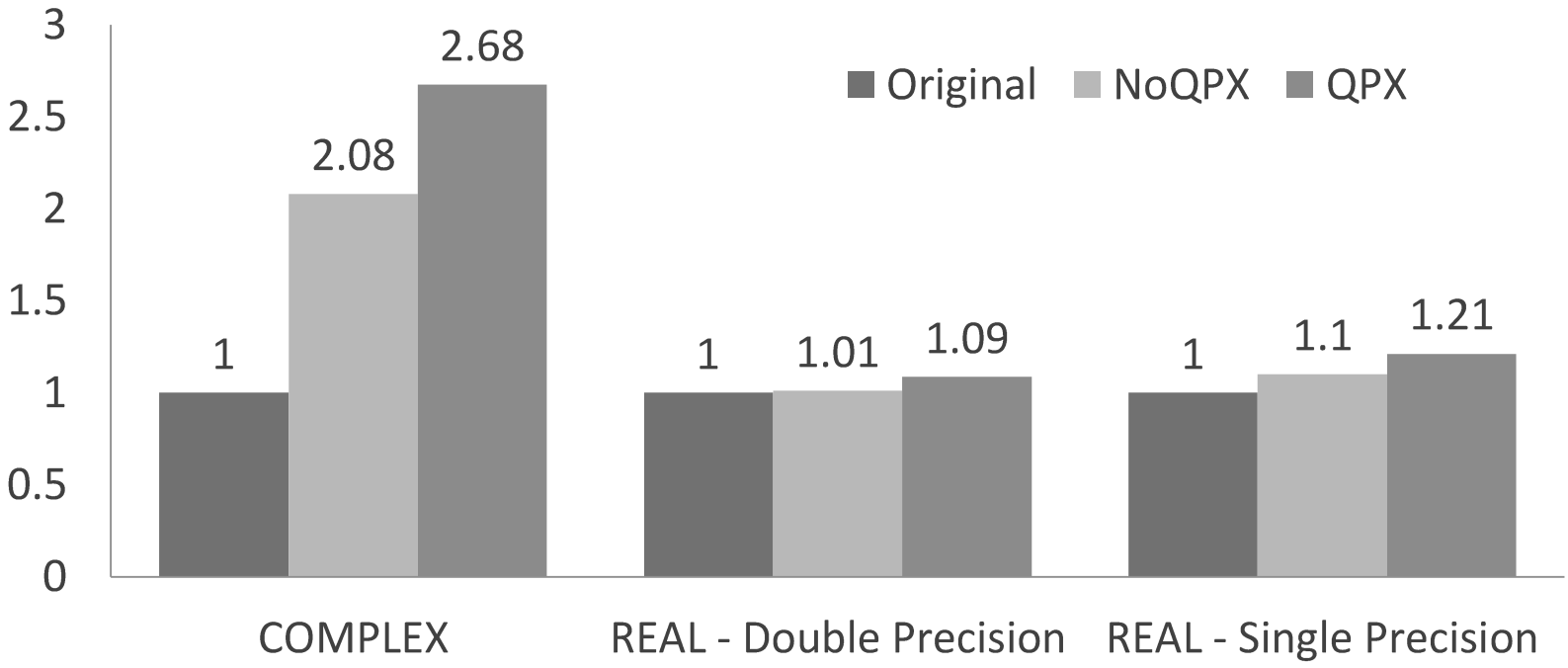
```
 8.581.366.867.332   All XU Instruction
 4.896.512.230.816   All AXU Instruction
13.017.533.928.058   FP Operations Group 1

Derived metrics for code block "mpiAll"
averaged over process(es) on node <0,0,0,0,0>:
Instruction mix:  FPU = 36.33 %,  FXU = 63.67 %
Instructions per cycle completed per core =
0.4417
Per cent of max issue rate per core = 28.12 %
Total weighted GFlops for this node = 10.922
Loads that hit in L1 d-cache =  88.60 %
                 L1P buffer =   5.92 %
                 L2 cache   =   4.50 %
                 DDR        =   0.98 %
DDR traffic for the node: ld = 3.503, st =
1.101, total = 4.604 (Bytes/cycle)
```

Percentage of peak= 5.33%

Total run time Speedup of 2.68 times

QMCPACK – performance on Blue Gene/Q



Application speedup using QPX and prefetching is **2.68** folds from original Algorithm.

QMCPACK – moving forward

- On Blue Gene/Q and Intel KNL:
  - Working on Nested OpenMp Parallelism to handle large QMC problems and take advantage of largenumber of cores
  - Better memory hierarchy and usage

- Challenges:
  - Percentage of peak is very low ~ 5-6%
    - Many kernels bandwidth limited (i.e. QPX will not help)
  - More compact representations of the wave function needed
    - Distribute/tessellate wave function without a performance hit

Thank you