Argonne NATIONAL LABORATORY

Application Performance Case Study

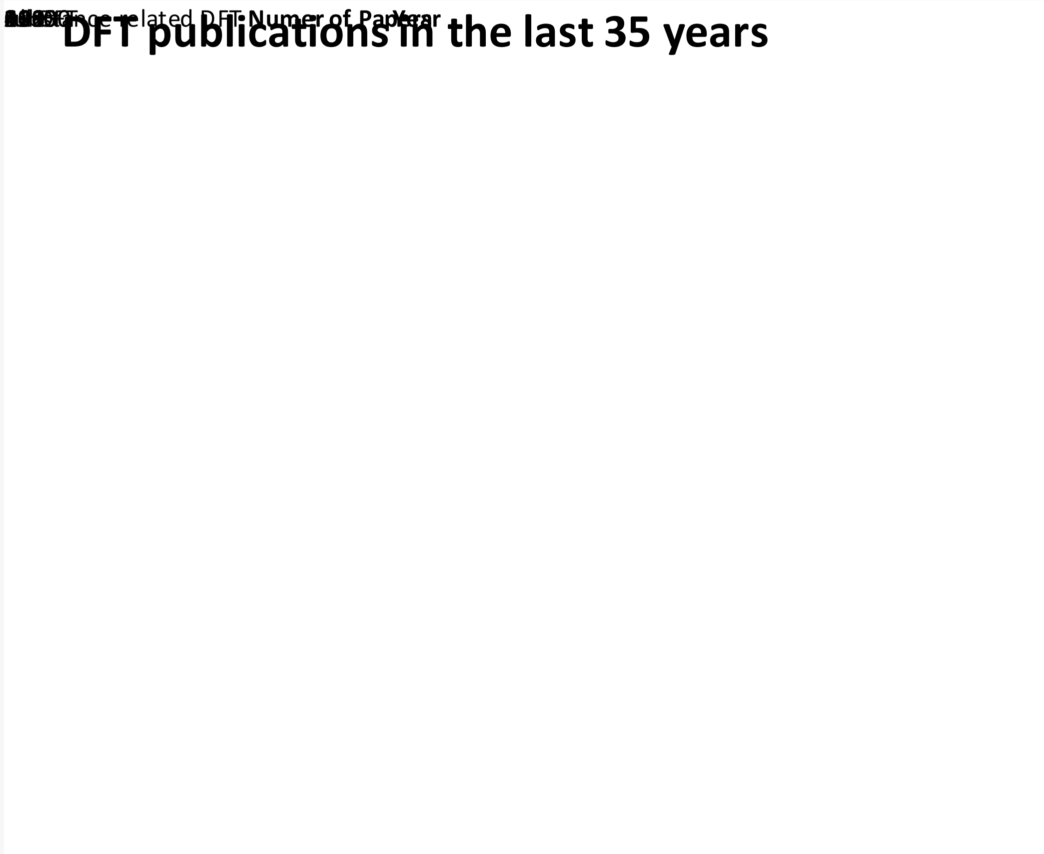
Enabling Large Scale DFT Simulations in KNL

Computational Performance Workshop
May 1, 2019

Álvaro V Mayagoitia
ALCF

Acknowledgement

ALCF Acknowledgement
This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.



Total publications as of 2015:
114K papers

Largest carbon calculations
C6000 PCCP Noel Y (2014)
C5120 PRB Holec D (2010)
C4860 PCCP Noel Y (2014)

Haunschild et al. *J Cheminform* (2016) **8**:52

http://www.crexplorer.net

Density Functional Theory

DFT is an exact theory, where the total energy of the system is described as functional of the charge density.

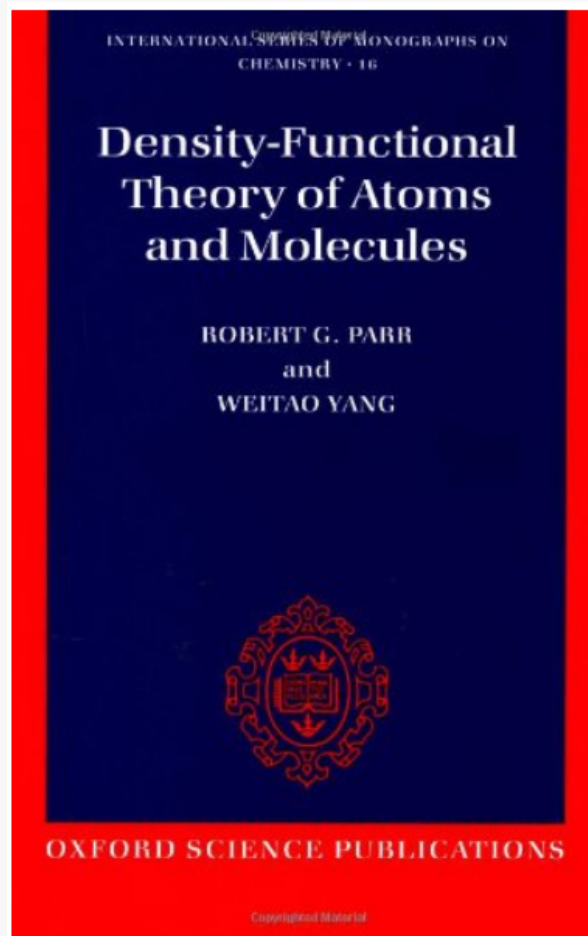
$\rho\left(\vec{r}\right)$ 3D function

$$E_{Total} = E[\rho\left(\vec{r}\right)]$$

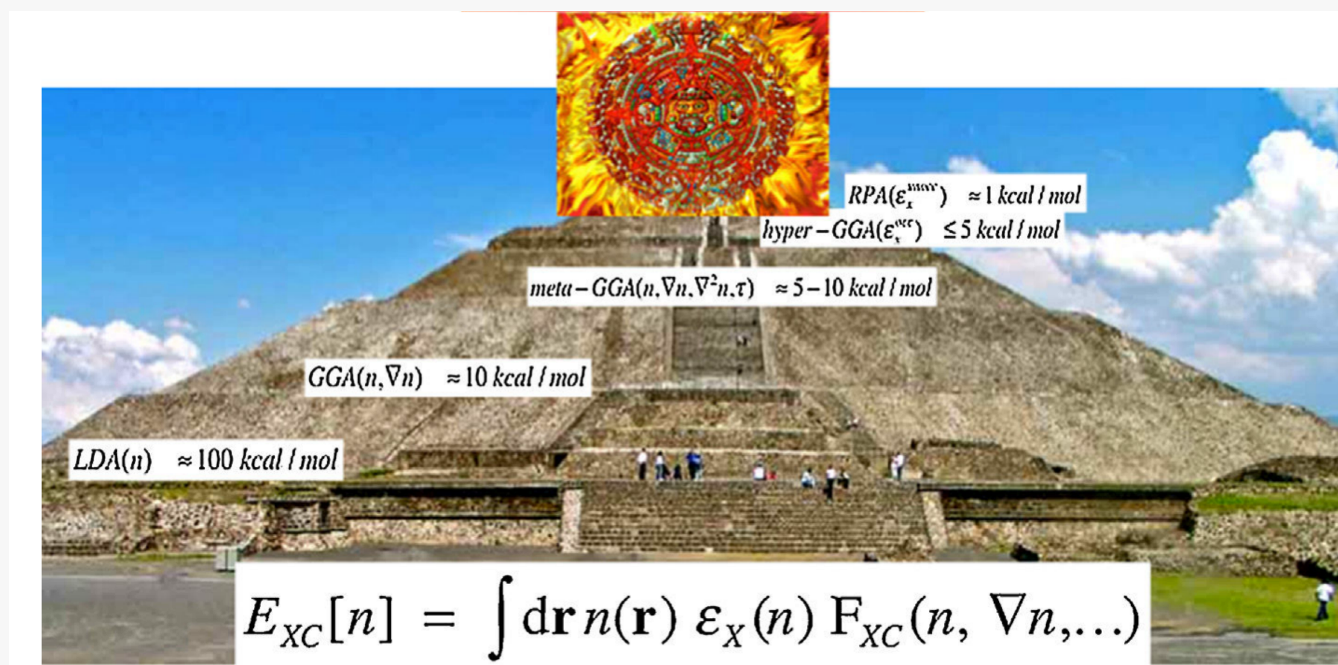
In the Kohn-Sham scheme the actual ground state energy is exact if the exchange correlation functional is known.

$$E[\rho(\vec{r})] = T_s[\rho] + J[\rho] + E_{xc}[\rho] + V_{ne}[\rho].$$

$$E_{xc}[\rho] = T[\rho] - T_s[\rho] + V_{ee}[\rho] - J[\rho]$$

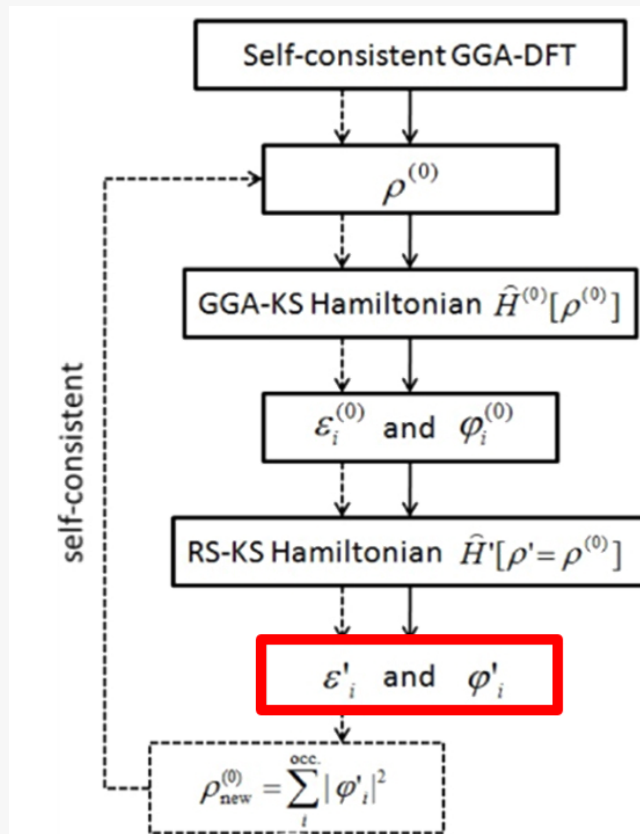


Jacob’s ladder – Aztec version



Author: A. Vela CINVESTAV

Kohn-Sham scheme



Density Functional Theory

$$E_{Total} = E[\rho(\vec{r})]$$

Auxiliary KS function

$$\Psi^{KS} = \sqrt{N!} det|\varphi_1(\vec{r_1})...\varphi_N(\vec{r_B})|$$

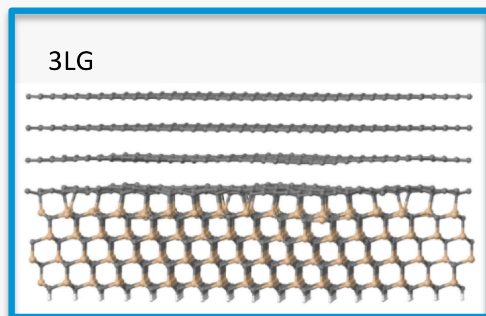
Density as sum of orbital densities

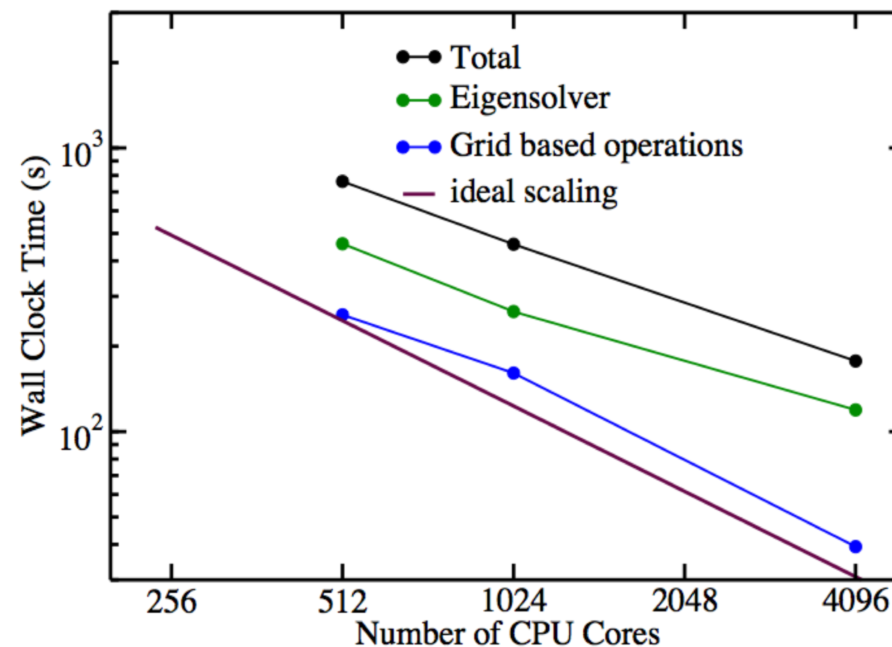
$$\rho(\vec{r}) = \sum_i |\varphi_i(\vec{r})|^2$$

Differential problem is now an Eigenvalue problem

2nd KS theorem:
Exact ground state density minimizes *E*

3LG:
3 layer graphene on SiC
88044 basis functions
2324 atoms
FHI-aims code



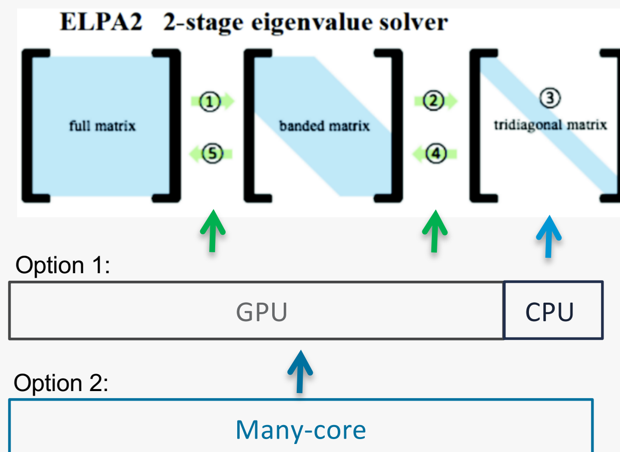


*Author: Volker Blum's team @Duke

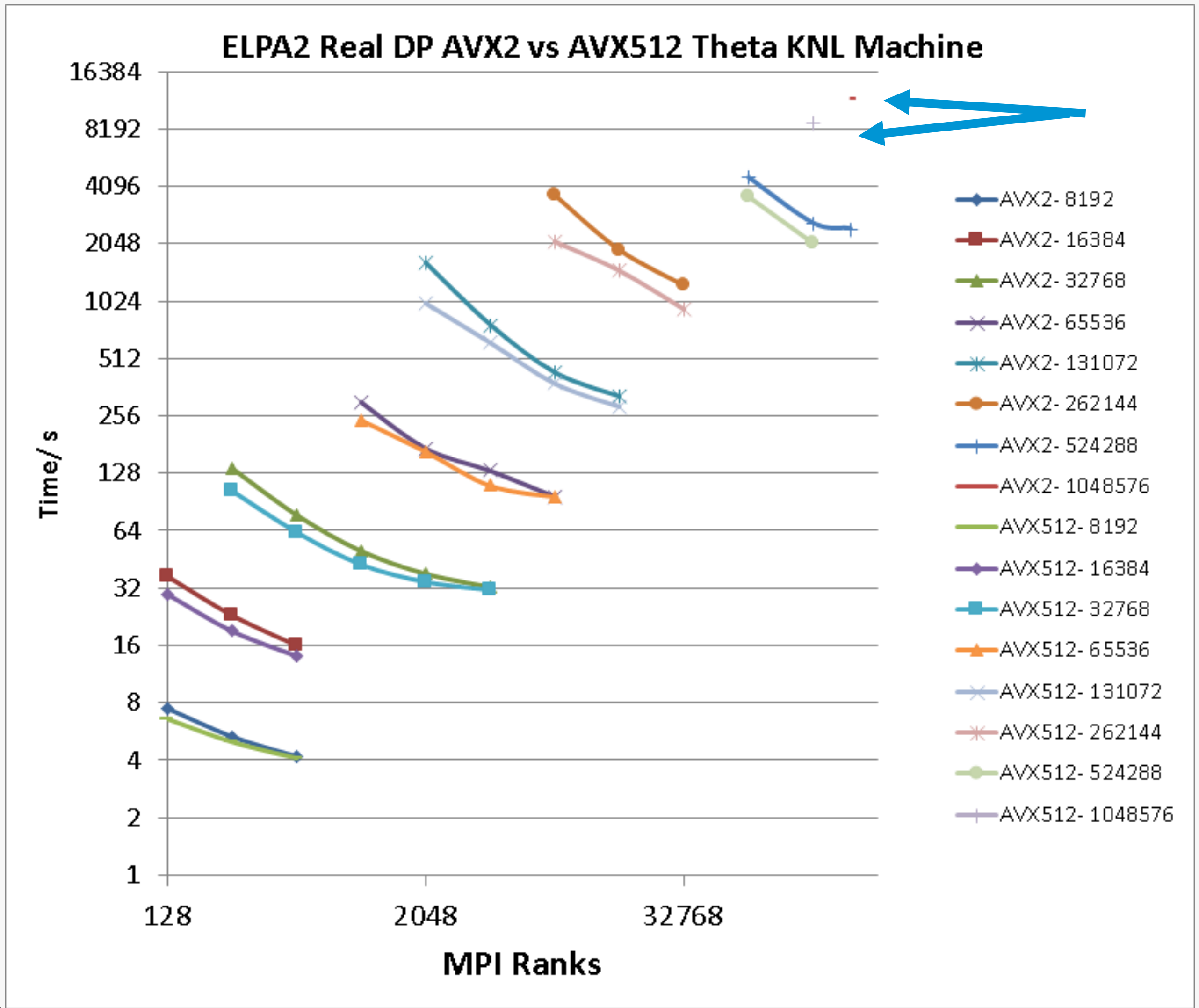
ELPA/ELSI

- Fast dense eigenvalue solver. Designed for high-performance architectures.
- Useful for electronic structure and many other applications. GPL license. F90 code.
- Uses a Block-cyclic distribution scheme mapping the group processors in a 2D grid. Relies on BLACS matrix parallelization layout.
- Uses efficiently external algebra libraries as BLAS, MKL, ESSL, etc.
- Main kernels in QR algorithms are hardcoded with intrinsic processor instructions (QPX, SSE, AVX, **AVX512**, etc).
- Solves for real or complex, and for entire or partial number of eigenvalues

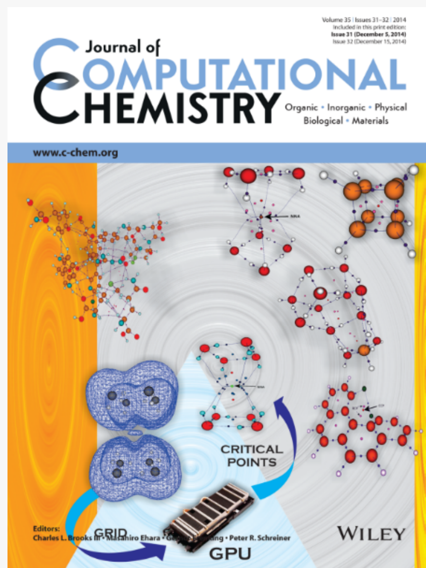
Support for:



Marek et al, *The Journal of Physics: Condensed Matter* 26, 213201 (2014)

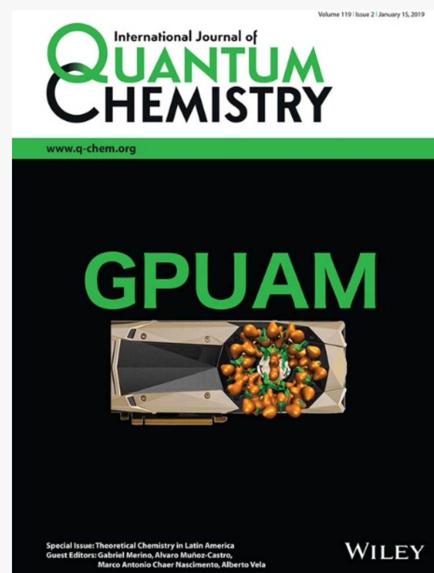


Theta (Cray/Intel) KNL-7210,64 MPI ranks per node, flatquad mode, Numa preferred MCRAM, AVX2 kernels, OpenMP=1

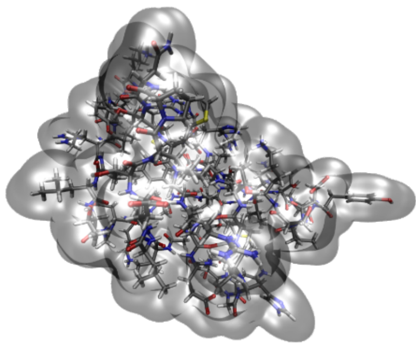


Grid-based algorithm to search critical points, in the electron density, accelerated by GPU

GPUs as boosters to analyze scalar and vector fields in quantum chemistry

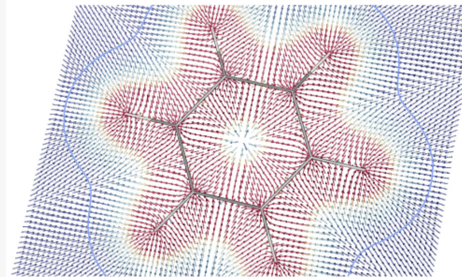


Density properties



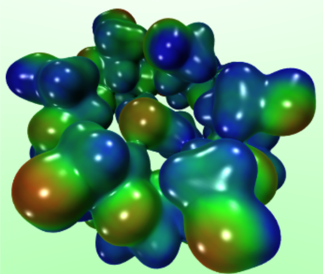
Electron density

$$\rho(\vec{r}) = \sum_{i}^{N} \omega_i \phi_i(\vec{r}) \phi_i(\vec{r})$$



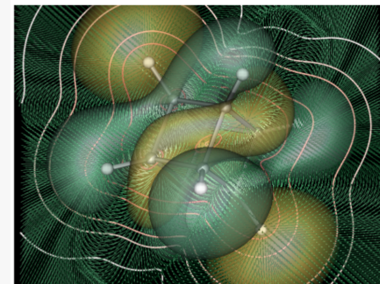
Density gradient

$$\nabla\rho(\vec{r}) = \hat{\imath}\frac{\partial\rho(\vec{r})}{\partial x} + \hat{\jmath}\frac{\partial\rho(\vec{r})}{\partial y} + \hat{k}\frac{\partial\rho(\vec{r})}{\partial z}$$



Electrostatic potential

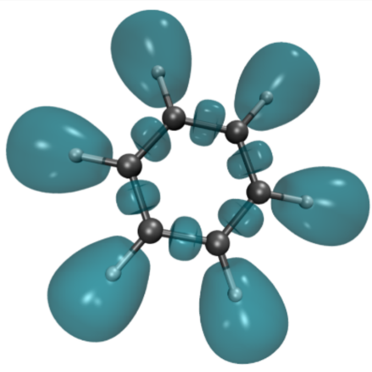
$$V(\vec{r}) = \sum_{A}^{M} \frac{Z_A}{|\vec{r} - \vec{A}|} - \int d\vec{r}\,' \frac{\rho(\vec{r}\,')}{|\vec{r} - \vec{r}\,'|}$$



Orbital density and gradients

$$\nabla\phi_i(\vec{r}) = \hat{\imath}\frac{\partial\phi_i(\vec{r})}{\partial x} + \hat{\jmath}\frac{\partial\phi_i(\vec{r})}{\partial y} + \hat{k}\frac{\partial\phi_i(\vec{r})}{\partial z}$$

Density properties

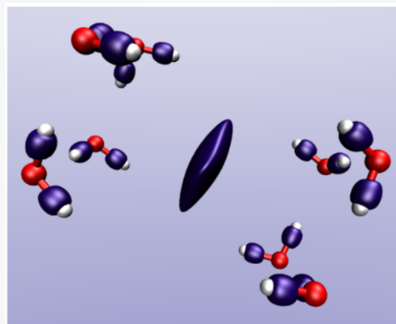


Electron localization function

$$\mathrm{ELF}(\vec{r}) = \frac{1}{1+\left(\frac{D(\vec{r})}{D_0(\vec{r})}\right)^2}$$

$$D_0(\vec{r}) = \frac{3}{10}\left(3\pi^2\right)^{2/3}\rho^{5/3}(\vec{r})$$

$$D(\vec{r}) = \frac{1}{2}\sum_i^N \omega_i\left|\nabla\phi_i(\vec{r})\right|^2 - \frac{1}{8}\frac{\left|\nabla\rho(\vec{r})\right|^2}{\rho(\vec{r})}$$

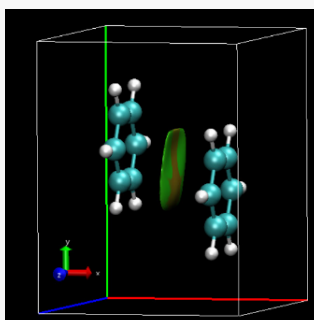


Electron localization function

$$\mathrm{LOL}(\vec{r}) = \frac{\tau(\vec{r})}{1+\tau(\vec{r})}$$

$$\tau(\vec{r}) = \frac{D_0(\vec{r})}{\frac{1}{2}\sum_i^N \omega_i\left|\nabla\phi_i(\vec{r})\right|^2}$$

$$D_0(\vec{r}) = \frac{3}{10}\left(3\pi^2\right)^{2/3}\rho^{5/3}(\vec{r})$$



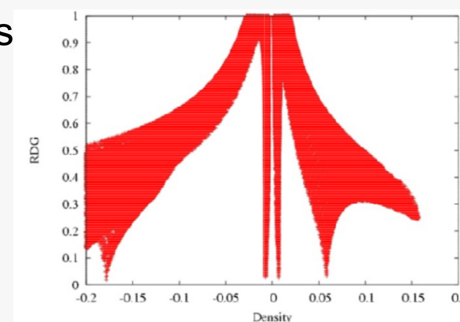
Non-covalent Interactions

Density

$$\rho(\vec{r}) = \sum_i^N \omega_i\phi_i(\vec{r})\phi_i(\vec{r})$$

Reduced density gradient

$$s(\vec{r}) = \frac{1}{2(3\pi^2)^{1/3}}\frac{\left|\nabla\rho(\vec{r})\right|}{\rho^{4/3}(\vec{r})}$$



Density as a fundamental variable

The charge density can be described exactly as a linear combination of natural orbitals

$$\rho(\vec{r}) = \sum_{i}^{N} \omega_i \left|\phi_i(\vec{r})\right|^2$$

The orbitals can be expressed in a function space as:

$$\phi_i(\vec{r}) = \sum_{\mu}^{K} c_{\mu i} g_{\mu}(\vec{r};\alpha,\vec{A},\vec{a})$$

Functions localized common in chemistry:

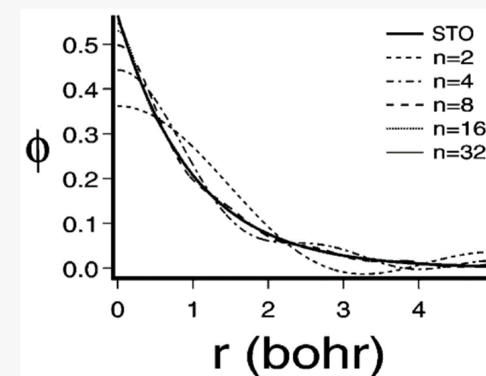
Gaussian type orbitals (GTO)

$$g_{\mu}(\vec{r};\alpha,\vec{A},\vec{a}) = (x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z}\exp\left(-\alpha\left|\vec{r}-\vec{A}\right|^2\right)$$

Slater type orbitals (STO)

$$f_{\mu}(\vec{r};n,\zeta,\vec{A},\vec{a}) = (x-A_x)^{a_x}(y-A_y)^{a_y}(z-A_z)^{a_z}\left|\vec{r}-\vec{A}\right|^{n-1}\exp\left(-\zeta\left|\vec{r}-\vec{A}\right|\right)$$

Orbital behavior



- (usually) More Gaussians functions means better description
- STO are more difficult to operate

Wavefunction in a file

- Most electronic structure programs can write wavefunction (WFN) files which contain the number of functions, atom positions, functions types, occupation number, etc.
- It is possible to convert between formats, for example: Molden to WFN (AIMS)
- WFN files are usually written in ASCII. They could be large files, it may be time consuming to read them.
- In this work we used Nwchem*

See Molden2AIM in github

https://sites.google.com/site/alvarovazquezmayagoitia/goals/codes/nwchem-notes/generator-of-aim-wavefunction-files-nwchem

Sample code

```
#pragma omp parallel shared(nomo,npri,vang,ctro,fcoor,fepri,fnocc,fcoef,    \
                            buffer,ptot,px,py,pz,h,x0,y0,z0)                \
                            private(i,ix,iy,iz,x,y,z)
{
#pragma omp parallel for
    for(i=0; i<ptot ;i++){
      ix = i/(py*pz);  iy = i%(py*pz)/pz;  iz = i%pz;
      x = x0 + ix*h ;  y = y0 + iy*h;  z = z0 + iz*h;

      buffer[i] = (*func)(x,y,z,nomo,npri,vang,
                          ctro,fcoor,fepri,fnocc,fcoef);
    }
}
  }
```

- Computation strategy 'similar' to GPU code reduces memory footprint

A sample of original code

- Conditional statements added to "skip calculations" contribute to prevent further vectorization
- Declarations in the beginning of the function make difficult to identify data dependency
- MKL math functions could be used if '-mkl' is present at compilation time
- Casting data types reduce loop efficiency
- Inner loop does the same computation all over and over
- Definition of the right types would remove unnecessary registers

```
float Cpurho (float x, float y, float z,int norb,int nprim,
              int *vang,int *ctro,float *coor,float *eprim,
              float *nocc,float *coef){

  int i,j,vj,centroj,pctro;
  int lx,ly,lz;
  float den, difx, dify, difz,mo, expo;
  float facx,facy,facz,rr;

  den = 0;
  for(i=0; i < norb; i++){
    mo = 0.0;
    pctro = -10;
    for(j=0; j < nprim; j++){
      vj = 3*j;

      if(ctro[j] != pctro){
        centroj = 3*ctro[j];

        difx = x - coor[centroj];
         dify = y - coor[centroj+1];
         difz = z - coor[centroj+2];

        rr = difx*difx + dify*dify + difz*difz;
      }

      expo = -eprim[j]*rr;
      if( expo >  CUTEXPO && fabs(coef[i*nprim+j]) >= CUTCOEF){
        expo = exp(expo)*coef[i*nprim+j];
        lx   = vang[vj];
        ly   = vang[vj+1];
        lz   = vang[vj+2];

        facx = PrePowCS(difx,lx);
        facy = PrePowCS(dify,ly);
        facz = PrePowCS(difz,lz);

        mo += facx*facy*facz*expo;
      }
      pctro = ctro[j];
    }
    den += (nocc[i]*mo*mo);
  }

  return den;
}
```

Hardware Specs

- Intel i7-6700 CPU @ 3.40GHz **
  - 4 cores, 8M cache
  - 24 GB RAM, 64 bit Ubuntu
- Intel(R) Xeon(R) CPU E7-8867v3 2.50GHz *
  - 16 core, 45 M cache
  - 1 TB RAM, 64bit CentOS
- Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz *
  - 18 cores, 45 M
  - 48GB, 64 bit CentOS
- Intel KNL 7210 *
  - 64 cores
  - 16 MCRAM
  - 192 DDR4, 64 bit CentOS
- Geforce Quadro K620 **
  - 384 CUDA Cores
  - 2 GB DDR3 memory
  - (attached to i7-6700)
- Tesla K80 x2 ***
  - 4992 CUDA cores
  - 24 GB of GDDR5 memory
  - 8.73 Tflop/s Nvida boost for SP
  - (attached Xeon E5-2620)

* JLSE.anl.gov
** Álvaro's desktop
*** Cooley

Tools used for development, analysis and debugging

- Suite Intel Parallel Studio XE 2017 Update 2 for Linux
  - Intel C/C++ Linux Compiler**
  - VTune Amplifer 2017
  - Advisor 2017
  - MKL
- Allinea 7 Forge
  - Map
  - DDT
- NVidia SDK 6.5

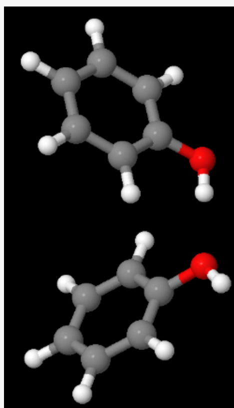
Intel Compilation flags

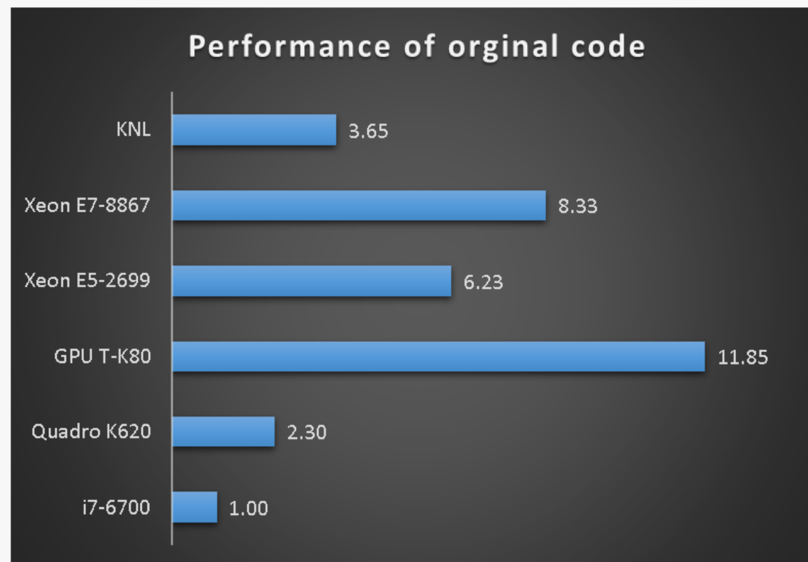
```
CFLAGS =  -g -O3 -fp-model fast=2 -align -p -mkl \
        -fimf-domain-exclusion=8 -qopenmp -qopt-report=5 \
        -qopt-report-phase=vec  -qopt-assume-safe-padding \
        -std=c99 \
        -xmic-avx512

LDFLAGS = -O3 -mkl -qopenmp
```

Executing the code

```
export KMP_AFFINITY="granularity=fine,scatter,noverbose"
export OMP_SCHEDULE=dynamic
export OMP_NUM_THREADS=256

numactl --membind 1 ./Gpuam.x < dens.in
```

https://software.intel.com/en-us/node/522691





Reference

Phenol dimer - 2(C6OH6)
Method : B3LYP/6-31G*
Primitives : 440
Total points : 872040

i7-6700: 8 threads
Xeon E5-2699: 72 threads
Xeon E7-8867: 64 threads
KNL: 256 threads

*In this example GPU uses multiple threads per point.
** time from wget_omp_time()

Fission of loops

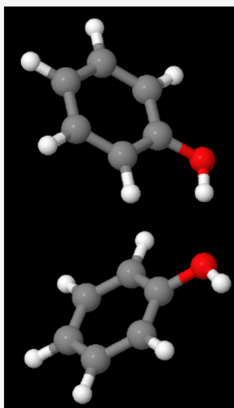
OLD

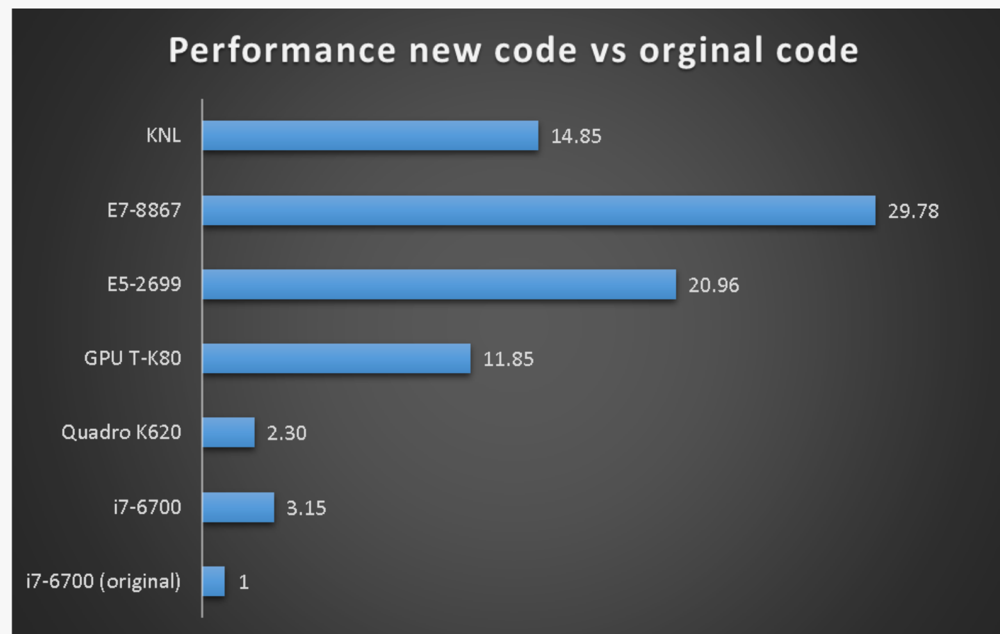
```
float Cpurho (float x, float y, float z,int norb,int nprim,
              int *vang,int *ctro,float *coor,float *eprim,
              float *nocc,float *coef){

  int i,j,vj,centroj,pctro;
  int lx,ly,lz;
  float den, difx, dify, difz,mo, expo;
  float facx,facy,facz,rr;

  den = 0.f;
  for(i=0; i < norb; i++){
    mo = 0.0;
    pctro = -10;
    for(j=0; j < nprim; j++){
      vj = 3*j;
      if(ctro[j] != pctro){
        centroj = 3*ctro[j];

        difx = x - coor[centroj];
             dify = y - coor[centroj+1];
             difz = z - coor[centroj+2];

        rr = difx*difx + dify*dify + difz*difz;
      }
      expo = -eprim[j]*rr;
      if( expo >  CUTEXPO && fabs(coef[i*nprim+j]) >= CUTCOEF){
        expo = expf(expo)*coef[i*nprim+j];
        lx   = vang[vj];
        ly   = vang[vj+1];
        lz   = vang[vj+2];

        facx = powf(difx,lx);
        facy = powf(dify,ly);
        facz = powf(difz,lz);
        mo += facx*facy*facz*expo;
      }
      pctro = ctro[j];
    }
    den += (nocc[i]*mo*mo);
  }

  return den;
}
```

NEW

```
float Cpurho (float x, float y, float z,int norb,int nprim,
              int *vang,int *ctro,float *coor,float *eprim,
              float *nocc,float *coef){

  float *moi;
  CreateArrayFlo(nprim, &moi, "MOI");
  for(int j=0; j < nprim; j++){
      const int vj = 3*j;

      const int centroj = 3*ctro[j];
      const float difx = x - coor[centroj];
      const float dify = y - coor[centroj+1];
      const float difz = z - coor[centroj+2];
      const float rr = difx*difx + dify*dify + difz*difz;
      const float expo = expf(-eprim[j]*rr);
      const float lx   = vang[vj];
      const float ly   = vang[vj+1];
      const float lz   = vang[vj+2];
      const float facx = powf(difx,lx);
      const float facy = powf(dify,ly);
      const float facz = powf(difz,lz);

      moi[j] = facx*facy*facz*expo;
  }

  float den = 0.f;
  for(int i=0; i < norb; i++){
     float mo=0.0f;
     const int i_prim= i*nprim;
     for(int j=0;j<nprim;j++){
        const float prim = moi[j]*coef[i_prim+j];
        mo+= prim;
     }
        den += (nocc[i]*mo*mo);
  }
  _mm_free(moi);
  return den;
}
```

New code comparison





Phenol dimer - 2(C6OH6)
Method : B3LYP/6-31G*
Primitives : 440
Total points : 872040

i7-6700: 8 threads
Xeon E5-2699: 72 threads
Xeon E7-8867: 64 threads

* More is better

Enabling memory alignment

Before

```
int CreateArrayFlo ( int n, float **ptr, char const *mess){
  //..
  *ptr = (float*) malloc( n*sizeof(float));
//..
   return 0;
}
```

After

```
int CreateArrayFlo ( int n, float **ptr, char const *mess){
//..
  int al = 64;
  *ptr = (float*) _mm_malloc( n*sizeof(float),al);
//..
   return 0;
}
```

See:
https://software.intel.com/en-us/articles/coding-for-performance-data-alignment-and-structures

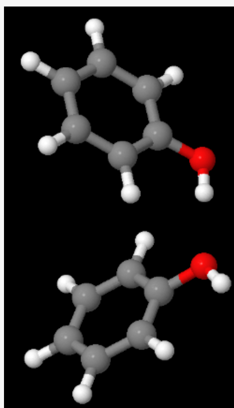
SIMD code

```
float Cpurho (float x, float y, float z,int norb,int nprim,
              int *vang,int *ctro,float *coor,float *eprim,
              float *nocc,float *coef){
  float *moi;
  CreateArrayFlo(nprim, &moi, "MOI");
 __assume_aligned(moi,64);
 __assume_aligned(ctro,64);
 __assume_aligned(vang, 64) ;
 __assume_aligned(coor,64);
 __assume_aligned(eprim,64);
 __assume_aligned(nocc,64);
 __assume_aligned(coef,64);
#pragma simd
  for(int j=0; j < nprim; j++){
      const int vj = 3*j;

      const int centroj = 3*ctro[j];
      const float difx = x - coor[centroj];
      const float dify = y - coor[centroj+1];
      const float difz = z - coor[centroj+2];
      const float rr = difx*difx + dify*dify + difz*difz;

      const float expo = expf(-eprim[j]*rr);
      const float lx   = vang[vj];
      const float ly   = vang[vj+1];
      const float lz   = vang[vj+2];
      const float facx = powf(difx,lx);
      const float facy = powf(dify,ly);
      const float facz = powf(difz,lz);

      moi[j] = facx*facy*facz*expo;
  }
  float den = 0.f;
  for(int i=0; i < norb; i++){
     float mo=0.0f;
     const int i_prim= i*nprim;
#pragma simd
     for(int j=0;j<nprim;j++){
        const float prim = moi[j]*coef[i_prim+j];
        mo+= prim;
     }
        den += (nocc[i]*mo*mo);
  }
  _mm_free(moi);

  return den;
}
```

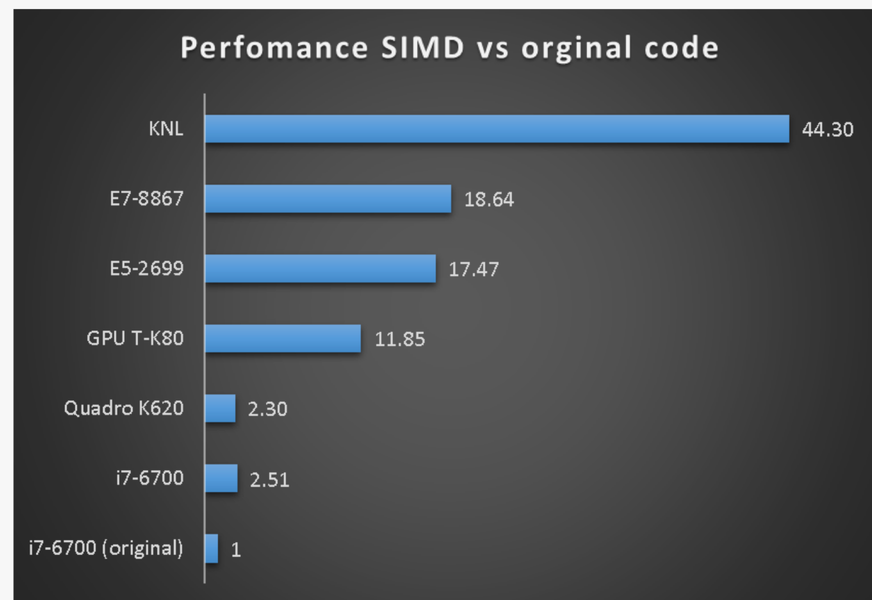
- This function is nested in an OpenMP region. The number of cycles in the loop is not large enough to pay the price of nesting threads
- When arrays were declared elsewhere, __assume_aligment should be present when calling SIMD to remove loop peeling of remains
- Use explicit functions corresponding to the data type, for example: powf(), expf() , etc

Argonne NATIONAL LABORATORY

SIMD code comparison



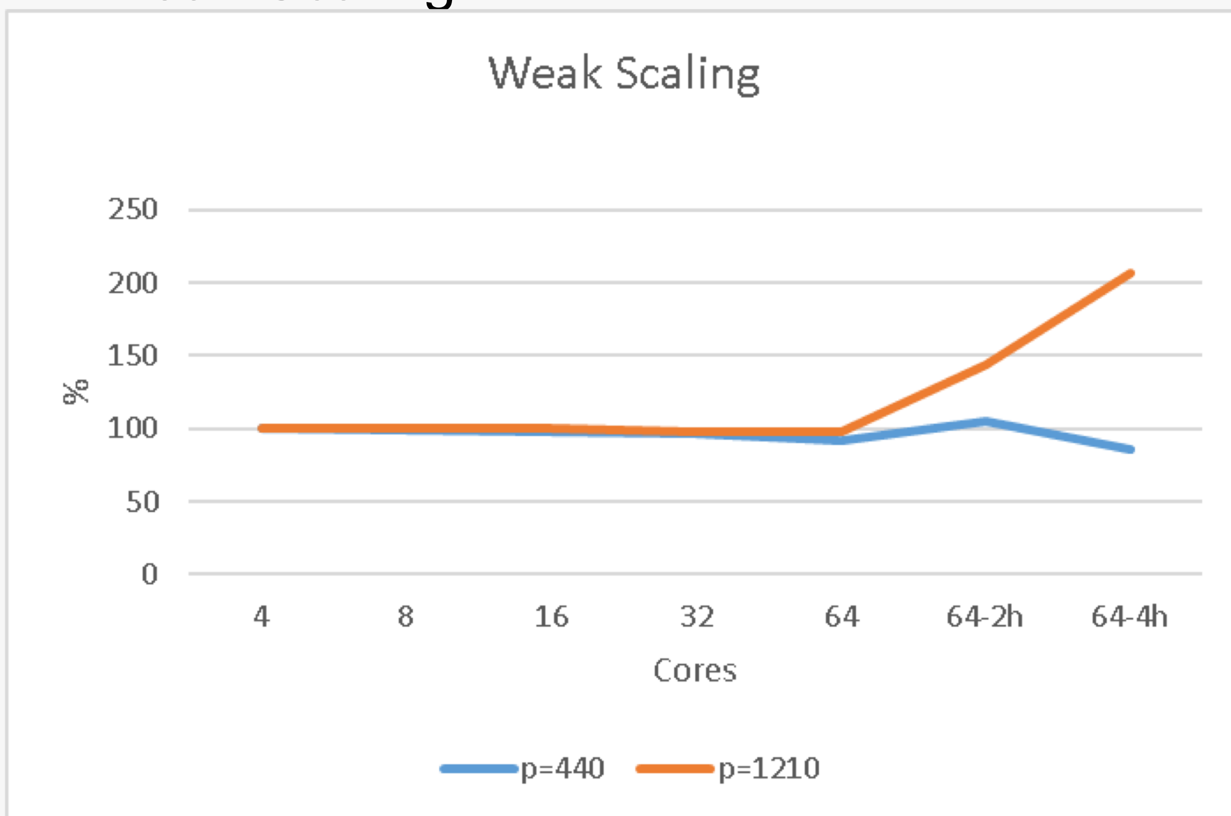
Phenol dimer - 2(C6OH6)
Method : B3LYP/6-31G*
Primitives : 440
Total points : 872040



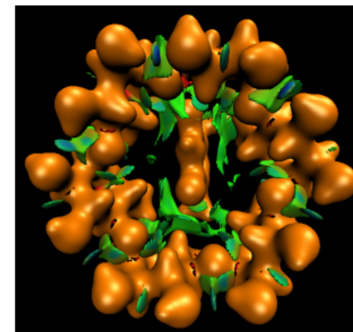
i7-6700: 8 threads
Xeon E5-2699: 72 threads
Xeon E7-8867: 64 threads

* More is better

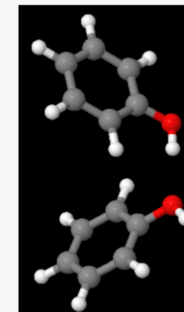
Weak Scaling in KNL



Ciclodextrine



Phenol



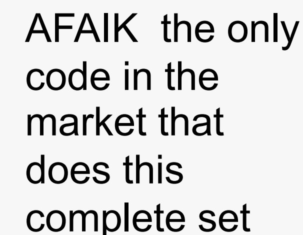
p = primitive functions

*More is better
**Also more points

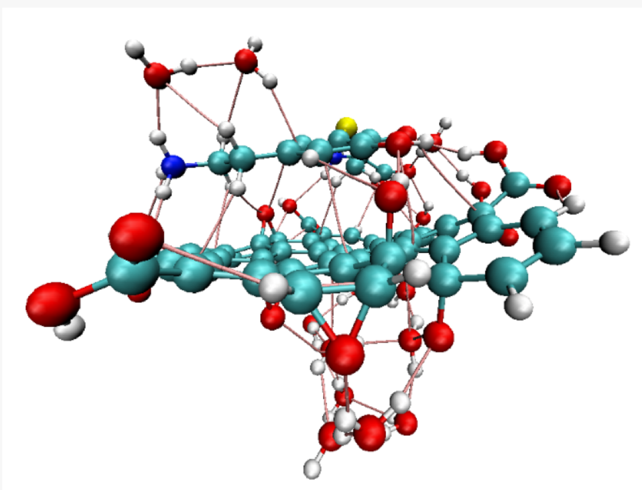
Routines optimized for KNL

Scalar and vector fields optimized:

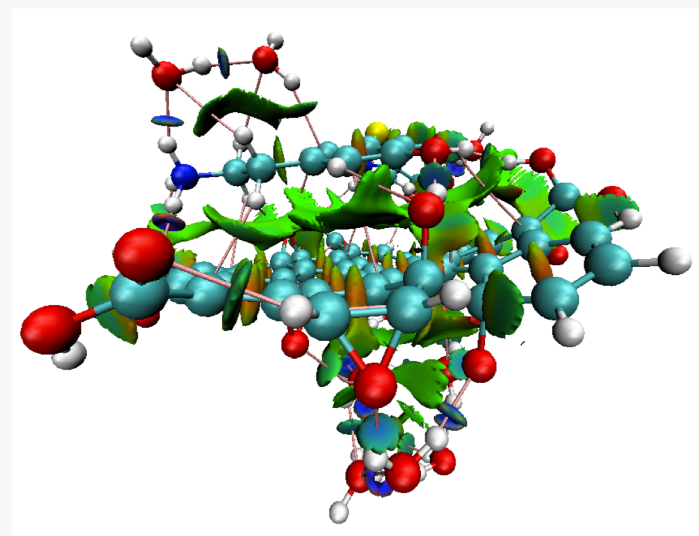
- **GTOs**
  - Molecular orbital
  - Electron density
  - Gradient and Laplacian of molecular orbitals and electron density
  - Kinetic energy density
  - Electron localization function
  - Non-covalent interaction index
  - Localized orbital locator
  - Search of critical points for electron density
  - Search of critical points for Laplacian of electron density
  - Electrostatic potential.
- **STOs**
  - Molecular orbital
  - Electron density
  - Gradient and Laplacian of molecular orbitals and electron density
  - Non-covalent interaction index
  - Search of critical points for electron density.



Application examples



Bond paths

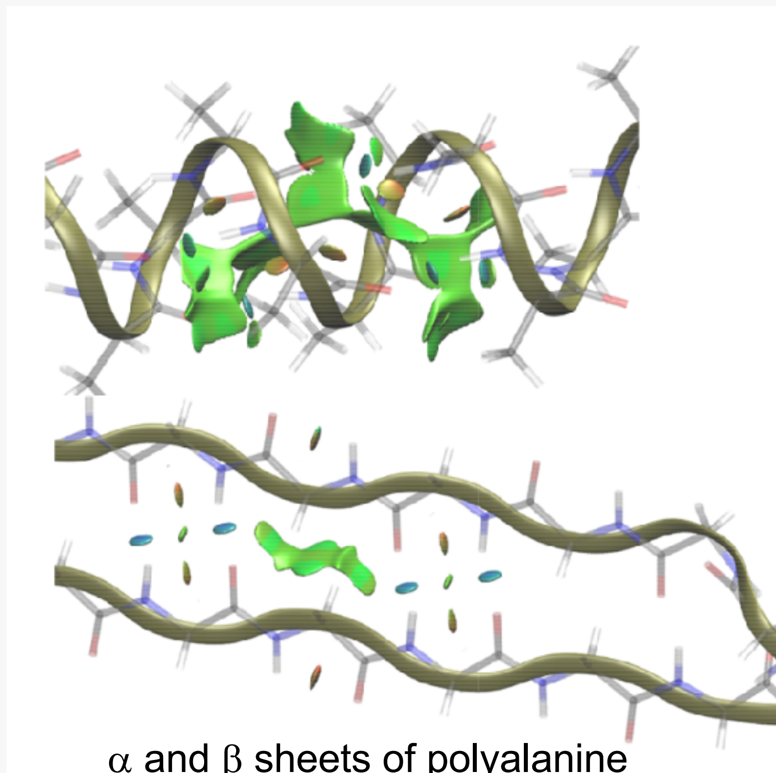


Bond paths and non-covalent interaction surfaces

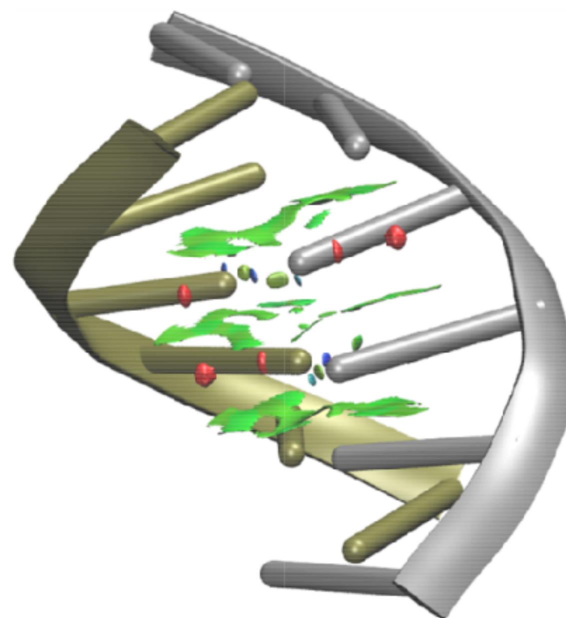
*10M points and 4k GTO functions
** Results in minutes instead hours

Author: Jorge Garza @UAM

Application examples



α and β sheets of polyalanine



Double helix DNA strands

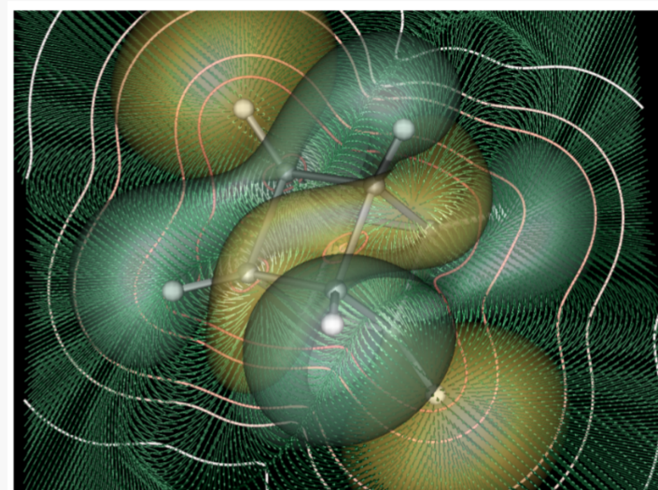
*Author: nciplot team

Conclusions and Insights

- Reusing allocated memory space was key to beat accelerators
- Intensive workload per thread and vectorization matters in KNL more than in other many-core processors
- SIMD instructions used Xeon not always give faster code. But in KNL, SIMD could make a big difference.
- OpenMP affinity should be properly chosen to use cores and hyper threading more efficiently

- KNL could be used as visualization processor for chemical properties

- Future work
  - When is possible replace loops for vector operations and use of tiles to optimally saturate registers. More loop fission.
  - Offer a library for quantum chemistry and visualization codes for on-the-fly evaluations
  - MPI version
  - Adoption of OpenMP 4.5 or OpenCL/SYCL (or other performance portability approach)



Q&A

Hardware specs

https://ark.intel.com/products/84681/Intel-Xeon-Processor-E7-8867-v3-45M-Cache-2_50-GHz

https://ark.intel.com/products/88196/Intel-Core-i7-6700-Processor-8M-Cache-up-to-4_00-GHz

https://ark.intel.com/products/81061/Intel-Xeon-Processor-E5-2699-v3-45M-Cache-2_30-GHz

https://ark.intel.com/products/94033/Intel-Xeon-Phi-Processor-7210-16GB-1_30-GHz-64-core

https://www.nvidia.com/object/tesla-k80.html

http://images.nvidia.com/content/pdf/quadro/data-sheets/75509_DS_NV_Quadro_K620_US_NV_HR.pdf

Argonne Leadership Computing Facility