# Walking to the Chapel: MADNESS - parallel runtime and application use cases

Robert J. Harrison

Institute for Advanced Computational Science
Stony Brook University

and

Center for Scientific Computing
Brookhaven National Laboratory

robert.harrison@stonybrook.edu



National Science Foundation
WHERE DISCOVERIES BEGIN

Stony Brook University

iACS
INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

# Molecular Science Software Project


**PNNL**
**Yuri Alexeev,**
**Eric Bylaska,**
**Bert deJong,**
**Mahin Hackler,**
**Karol Kowalski,**
**Lisa Pollack,**
**Tjerk Straatsma,**
**Marat Valiev,**
**Edo Apra**

**ISU and Ames**
**Theresa Windus**

**SBU & BNL**
**Robert Harrison**

**MS$^3$**
MOLECULAR SCIENCE SOFTWARE SUITE

**ECCE**
EXTENSIBLE COMPUTATIONAL CHEMISTRY ENVIRONMENT

**NWCHEM**
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

**GA TOOLS**
PARALLEL COMPUTING LIBRARIES AND SOFTWARE TOOLS

**(Jarek Nieplocha), Manoj Krishnan, Bruce Palmer, Daniel Chavarría, Sriram Krishnamoorthy**


**Gary Black,**
**Brett Didier,**
**Todd Elsenthagen,**
**Sue Havre,**
**Carina Lansing,**
**Bruce Palmer,**
**Karen Schuchardt,**
**Lisong Sun**
**Erich Vorpagel**

http://www.nwchem-sw.org

# Fock matrix in a Nutshell

$$F_{ij} = \sum_{kl} \left( 2(ij|kl) - (ik|jl) \right) D_{kl}$$

$$(\mu\nu|\sigma\lambda) = \int_{-\infty}^{\infty} g_\mu(r_1) g_\nu(r_1) \frac{1}{r_{12}} g_\sigma(r_2) g_\lambda(r_2) dr_1 dr_2$$

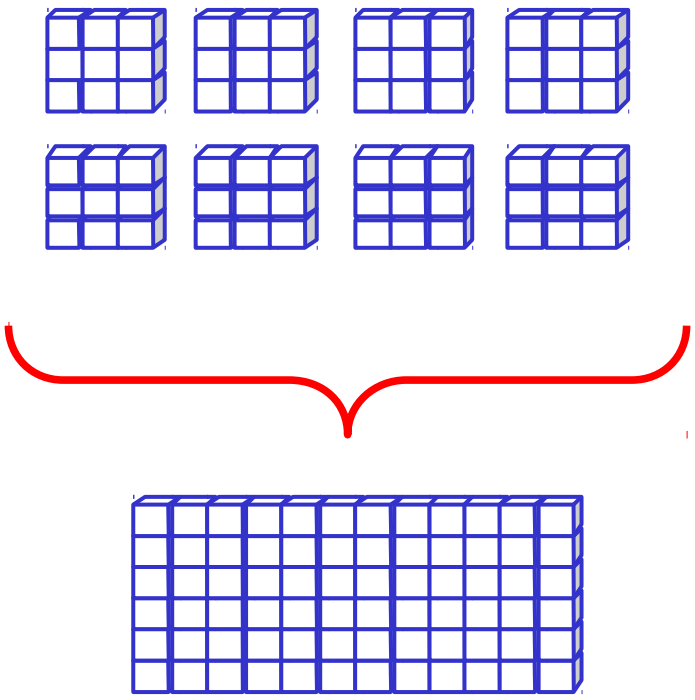
**1 integral contributes to 6 Fock Matrix elements**

$$(\mu\nu|\sigma\lambda) \otimes \begin{Bmatrix} D_{\mu\nu} \\ D_{\mu\sigma} \\ D_{\mu\lambda} \\ D_{\nu\sigma} \\ D_{\nu\lambda} \\ D_{\sigma\lambda} \end{Bmatrix} \Rightarrow \begin{Bmatrix} F_{\mu\nu} \\ F_{\mu\sigma} \\ F_{\mu\lambda} \\ F_{\nu\sigma} \\ F_{\nu\lambda} \\ F_{\sigma\lambda} \end{Bmatrix}$$

- Sparsity, variable integral costs, algorithm constraints, symmetry, shell blocking, ...

# Global Arrays (technologies)

Physically distributed data



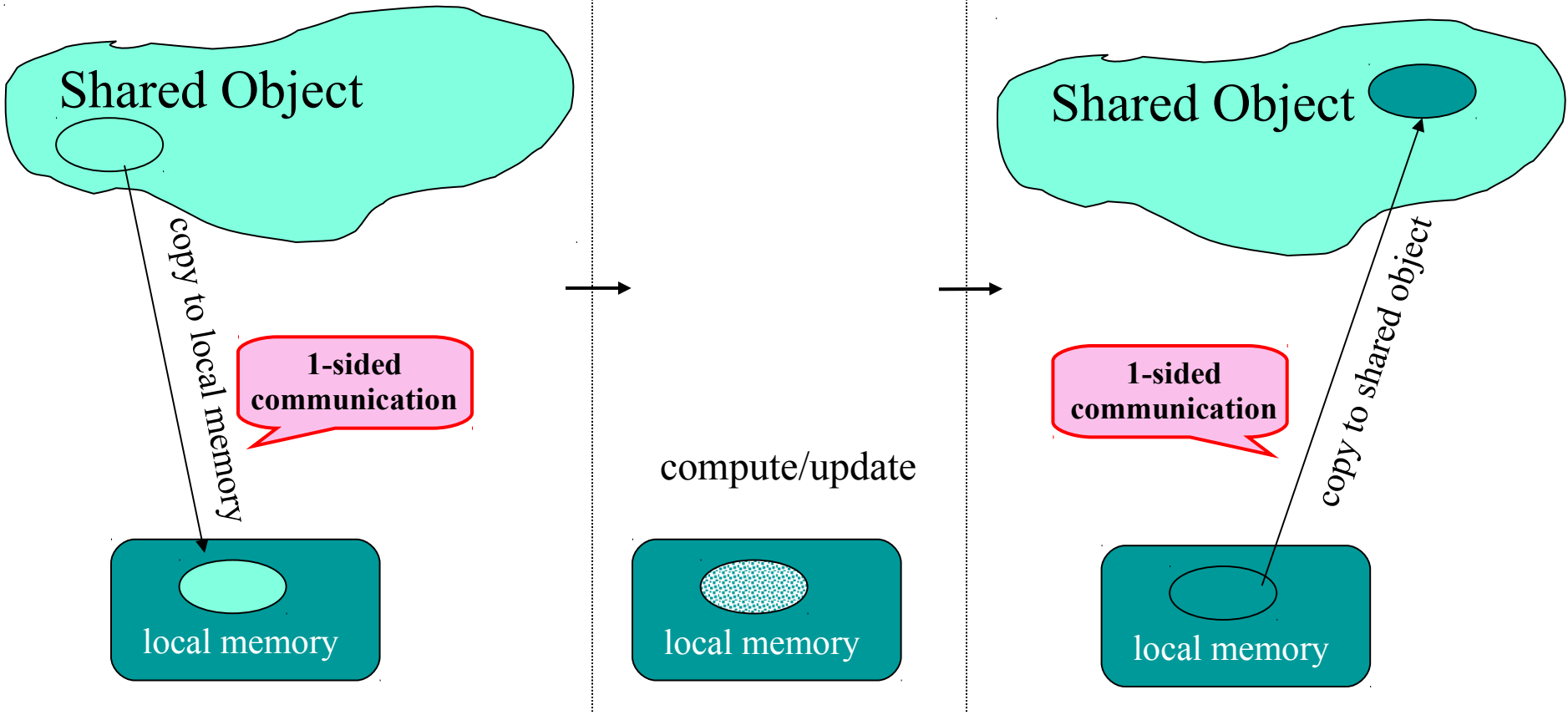
Single, shared data structure

- Shared-memory-like model
  - Fast local access
  - NUMA aware and easy to use
  - MIMD and data-parallel modes
  - Inter-operates with MPI, …
- BLAS and linear algebra interface
- Ported to major parallel machines
  - IBM, Cray, SGI, clusters,...
- Originated in an HPCC project
- Used by most major chemistry codes, financial futures forecasting, astrophysics, computer graphics
- Supported by DOE

- One of the legacies of Jarek Nieplocha, PNNL

http://www.emsl.pnl.gov/docs/global/

# Non-uniform memory access model of computation



# Distributed data SCF

- First success for NWChem and Global Arrays

```
do tiles of i
    do tiles of j
        do tiles of k
            do tiles of l
                get patches ij, ik, il, jk, jl, kl
                compute integrals
                accumulate results back into patches
```

Parallel loop nest

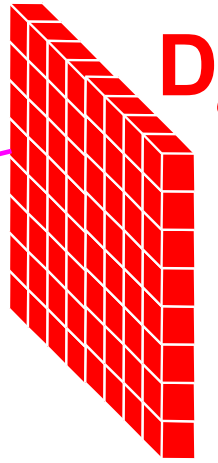
Mini-apps used to evaluate HPCS languages Chapel, X10, Fortress
- just the data flow

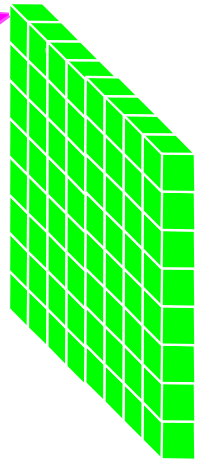
$B$ = block size

$$t_{\text{comm}} = O(B^2) \qquad t_{\text{compute}} = O(B^4) \qquad \frac{t_{\text{compute}}}{t_{\text{comm}}} = O(B^2)$$

# Dynamic load balancing

```
my_next_task = SharedCounter(chunksize)
do i=1,max_i
   if(i.eq.my_next_task) then
      call ga_get(                    )

            (do work)

      call ga_acc(                    )
   my_next_task = SharedCounter(chunksize)
   endif
enddo
Barrier()
```



# MADNESS

*Multiresolution*
*Adaptive*
*Numerical*
*Scientific*
*Simulation*

# Multiresolution Adaptive Numerical Scientific Simulation


*Robert J. Harrison[1], Scott Thornton[1], George I. Fann[2], Diego Galindo[2], Judy Hill[2], Jun Jia[2], Gregory Beylkin[4], Lucas Monzon[4], Hideo Sekino[5] Edward Valeev[6], Jeff Hammond[7], Nichols Romero[7], Alvaro Vasquez[7]*

*[1]Stony Brook University, Brookhaven National Laboratory*
*[2]Oak Ridge National Laboratory*
*[4]University of Colorado*
*[5]Toyohashi Technical University, Japan*
*[6]Virginia Tech*
*[7]Argonne National Laboratory*

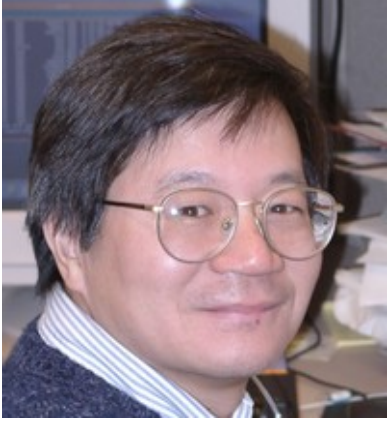
*robert.harrison@gmail.com*




BROOKHAVEN
NATIONAL LABORATORY


National Science Foundation
WHERE DISCOVERIES BEGIN

Stony Brook University



George Fann

Judy Hill

Gregory Beylkin

Rebecca Hartman-Baker

Jeff Hammond

Ariana Beste

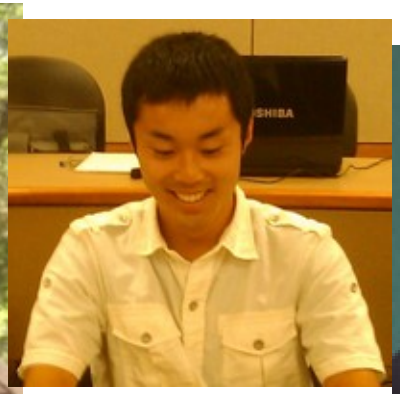
Eduard Valeyev

Alvaro Vasquez

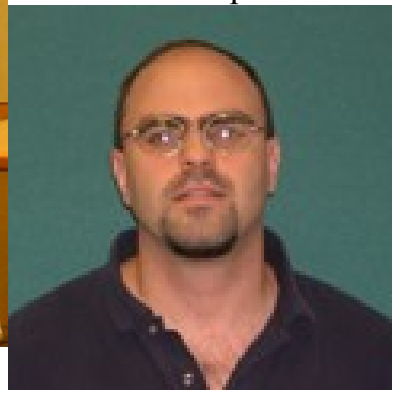
Hideo Sekino

Robert Harrison

Nicholas Vence



Takahiro Ii



Scott Thornton

Matt Reuter

Nichols Romero

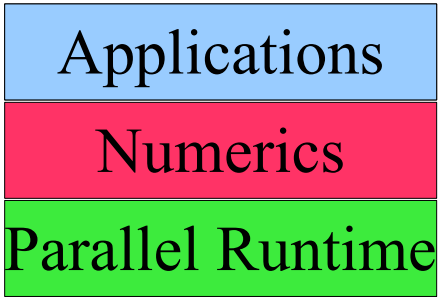


Jia, Kato, Calvin, Pei, ...

# What is MADNESS?

- A general purpose numerical environment for reliable and fast scientific simulation
  - Chemistry, nuclear physics, atomic physics, material science, nanoscience, climate, fusion, ...
- A general purpose parallel programming environment designed for the peta/exa-scales
- Addresses many of the sources of complexity that constrain our HPC ambitions

http://code.google.com/p/m-a-d-n-e-s-s
http://harrison2.chem.utk.edu/~rjh/madness/



# Why MADNESS?

- Reduces S/W complexity
  - MATLAB-like level of composition of scientific problems with guaranteed speed and precision
  - Programmer not responsible for managing dependencies, scheduling, or placement
- Reduces numerical complexity
  - Solution of integral not differential equations
  - Framework makes latest techniques in applied math and physics available to wide audience

# Big picture

- Want robust algorithms that scale correctly with system size and are easy to write
- Robust, accurate, fast computation
  - Gaussian basis sets: high accuracy yields dense matrices and linear dependence – $O(N^3)$
  - Plane waves: force pseudo-potentials – $O(N^3)$
  - $O(N \log^m N \log^k \varepsilon)$ is possible, guaranteed $\varepsilon$
- Semantic gap
  - Why are our equations just O(100) lines but programs O(1M) lines?
- Facile path from laptop to exaflop

# E.g., with guaranteed precision of 1e-6 form a numerical representation of a Gaussian in the cube $[-20,20]^3$, solve Poisson's equation, and plot the resulting potential (all running in parallel with threads+MPI)

Let

$$\Omega = [-20, 20]^3$$

$$\epsilon = 1e-6$$

$$g = x \rightarrow \exp\left(-\left(x_0^2 + x_1^2 + x_2^2\right)\right) * \pi^{-1.5}$$

In

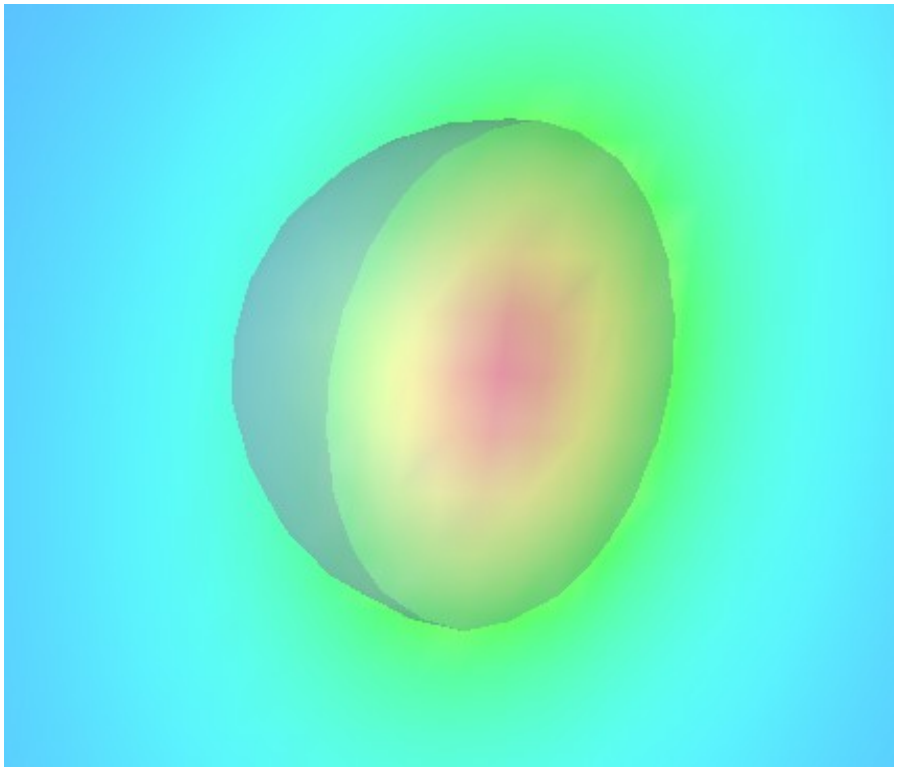
$$f = \mathcal{F}\, g$$

$$u = \nabla^{-2}\left(-4 * \pi * f\right)$$

print "norm of f", $\langle f \rangle$, "energy", $\langle f | u \rangle * 0.5$

plot $u$

End



*output:* norm of f 1.00000000e+00 energy 3.98920526e-01

There are only two lines doing real work. First the Gaussian (g) is projected into the adaptive basis to the default precision. Second, the Green's function is applied. The exact results are norm=1.0 and energy=0.3989422804.

Let

$$\Omega = [-20, 20]^3$$

$$r = x \to \sqrt{x_0^2 + x_1^2 + x_2^2}$$

$$g = x \to \exp(-2 * r(x))$$

$$v = x \to -\frac{2}{r(x)}$$

In

$$\nu = \mathcal{F}\, v$$

$$\phi = \mathcal{F}\, g$$

$$\lambda = -1.0$$

for $i \in [0, 10]$

$$\phi = \phi * \|\phi\|^{-1}$$

$$V = \nu - \nabla^{-2}\left(4 * \pi * \phi^2\right)$$

$$\psi = -2 * \left(-2 * \lambda - \nabla^2\right)^{-1}(V * \phi)$$

$$\lambda = \lambda + \frac{\langle V * \phi | \psi - \phi \rangle}{\langle \psi | \psi \rangle}$$

$$\phi = \psi$$

print "iter", $i$, "norm", $\|\phi\|$, "eval", $\lambda$

end

End

# He atom Hartree-Fock

Compose directly in terms of functions and operators

This is a Latex rendering of a program to solve the Hartree-Fock equations for the helium atom

The compiler also output a C++ code that can be compiled without modification and run in parallel

# “Fast” algorithms

- Fast in mathematical sense
  - Optimal scaling of cost with accuracy & size
- Multigrid method – Brandt (1977)
  - Iterative solution of differential equations
  - Analyzes solution/error at different length scales
- Fast multipole method – Greengard, Rokhlin (1987)
  - Fast application of dense operators
  - Exploits smoothness of operators
- Multiresolution analysis
  - Exploits smoothness of operators and functions

# The math behind the MADNESS

- Multiresolution

$$V_0 \subset V_1 \subset \cdots \subset V_n$$

$$V_n = V_0 + \left(V_1 - V_0\right) + \cdots + \left(V_n - V_{n-1}\right)$$

- Low-separation rank

$$f(x_1, \ldots, x_n) = \sum_{l=1}^{M} \sigma_l \prod_{i=1}^{d} f_i^{(l)}(x_i) + O(\epsilon)$$

$$\left\| f_i^{(l)} \right\|_2 = 1 \qquad \sigma_l > 0$$

- Low-operator rank

$$A = \sum_{\mu=1}^{r} u_\mu \, \sigma_\mu \, v_\mu^T + O(\epsilon)$$

$$\sigma_\mu > 0 \qquad v_\mu^T v_\lambda = u_\mu^T u_\lambda = \delta_{\mu\nu}$$

# How to "think" multiresolution

- Consider a ladder of function spaces

$$V_0 \subset V_1 \subset \cdots \subset V_n$$

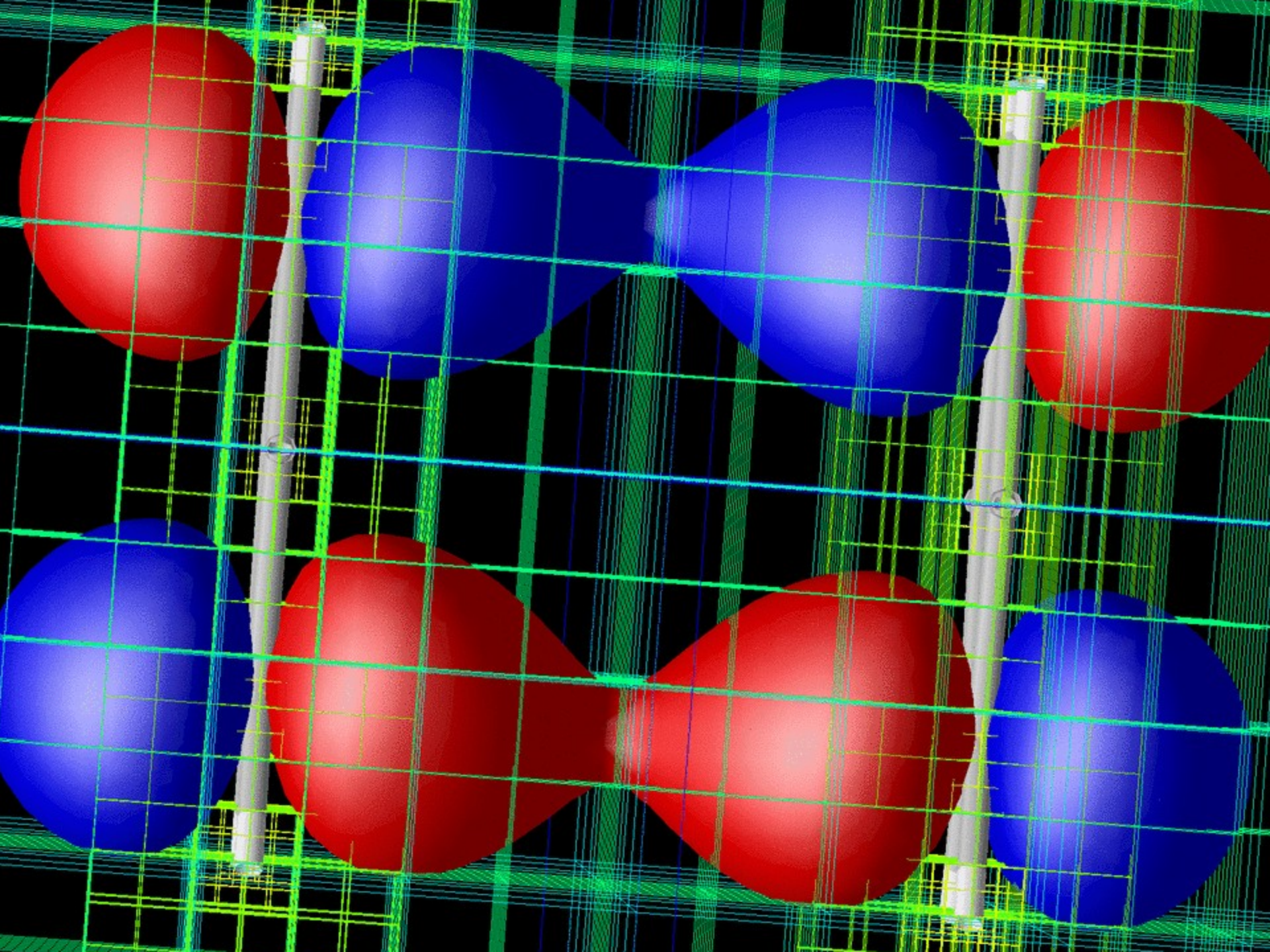
  - E.g., increasing quality atomic basis sets, or finer resolution grids, …
- Telescoping series

$$V_n = V_0 + \left(V_1 - V_0\right) + \cdots + \left(V_n - V_{n-1}\right)$$

  - Instead of using the most accurate representation, use the difference between successive approximations
  - Representation on $V_0$ small/dense; differences sparse
  - Computationally efficient; possible insights

# Why “think” multiresolution?

- It is everywhere in nature/chemistry/physics
  - Core/valence; high/low frequency; short/long range; smooth/non-smooth; atomic/nano/micro/macro scale
- Common to separate just two scales
  - E.g., core orbital heavily contracted, valence flexible
  - More efficient, compact, and numerically stable
- Multiresolution
  - Recursively separate all length/time scales
  - Computationally efficient and numerically stable
  - Coarse-scale models that capture fine-scale detail



# Example tree in Haar basis

Haar basis is a piecewise constant (like a histogram)

- Not useful for real calculations but easy to visualize and of fundamental importance

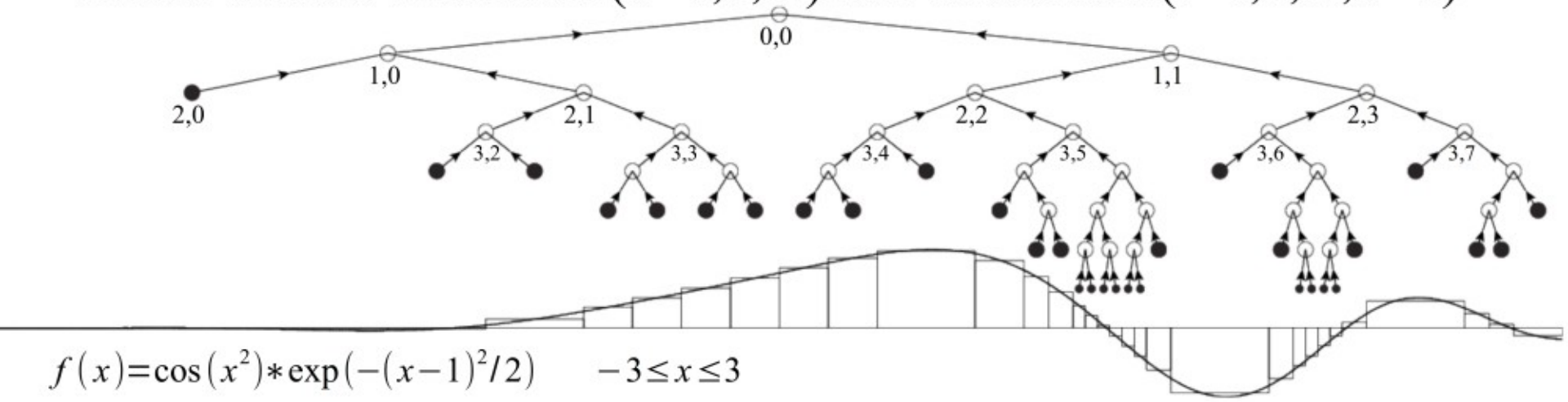
Adaptive local refinement until local error measure is satisfied

- Smaller boxes where rate of change is high (and value not negligible)

Conventional adaptive mesh corresponds to boxes

Construct tree connecting fine-scale to coarser-scale boxes

Boxes labeled with level ($n=0,1,\ldots$) and translation ($l=0,1,\ldots,2^n-1$)



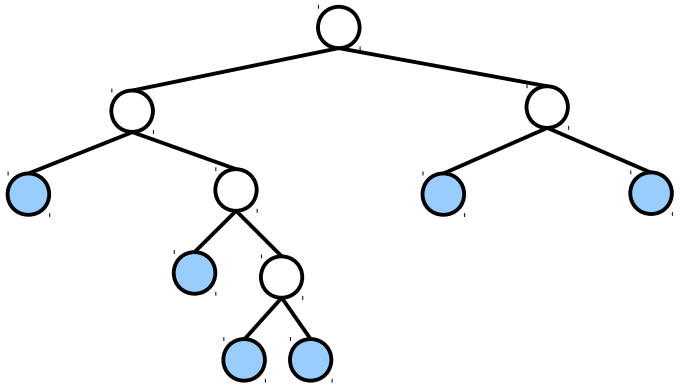

$$f(x)=\cos(x^2)*\exp(-(x-1)^2/2) \qquad -3\le x\le 3$$

# Another Key Component

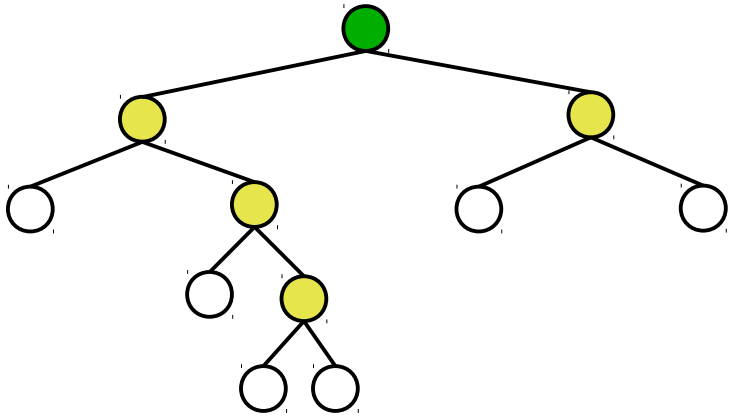
- Trade precision for speed – everywhere
  - Don't do anything exactly
  - Perform everything to $O(\varepsilon)$
  - Require
    - Robustness
    - Speed, and
    - Guaranteed, arbitrary, *finite* precision

# Please forget about wavelets

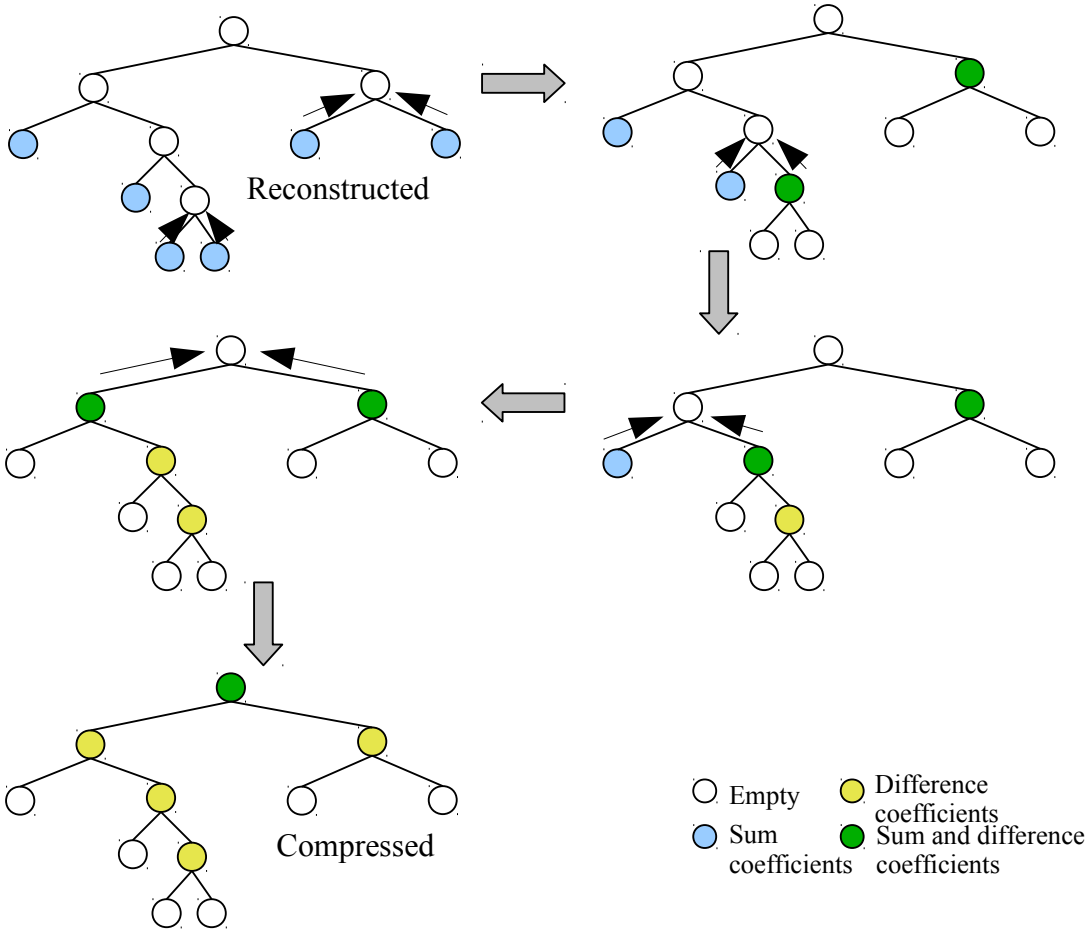
- They are not central
- Wavelets are a convenient basis for spanning $V_n$-$V_{n-1}$ and understanding its properties
- But you don't actually need to use them
  - MADNESS does still compute wavelet coefficients, but *Beylkin's new code does not*
- Please remember this …
  - Discontinuous spectral element with multi-resolution and separated representations for fast computation with guaranteed precision in many dimensions.



Tree in **reconstructed** form. Scaling function (sum) coefficients at leaf nodes. Interior nodes empty.



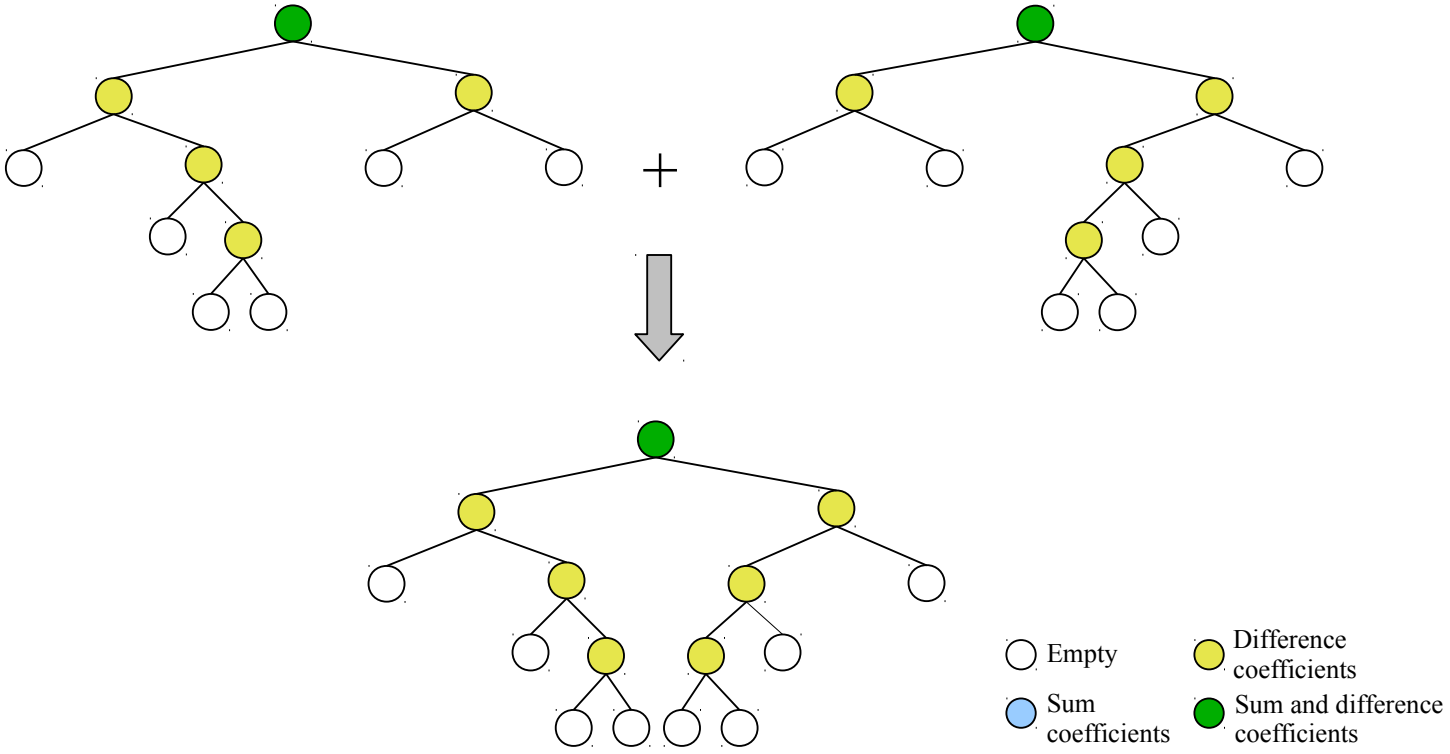
Tree in **compressed** form. Wavelet (difference) coefficients at interior nodes, with scaling functions coefficients also at root. Leaf nodes empty.



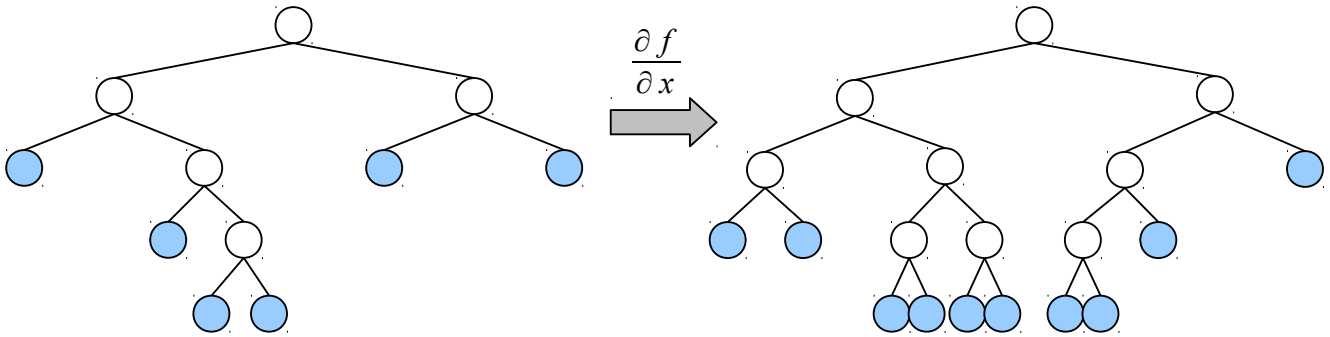

**Compression algorithm.** Starting from leaf nodes, scaling function (sum) coefficients are passed to parent. Parent "filters" the childrens' coefficients to produce sum and wavelet (difference) coefficients at that level, then passes sum coefficients to its parent.

Reconstruction is simply the reverse processes.

To produce the non-standard form the compression algorithm is run but scaling function coefficients are retained at the leaf and interior nodes.



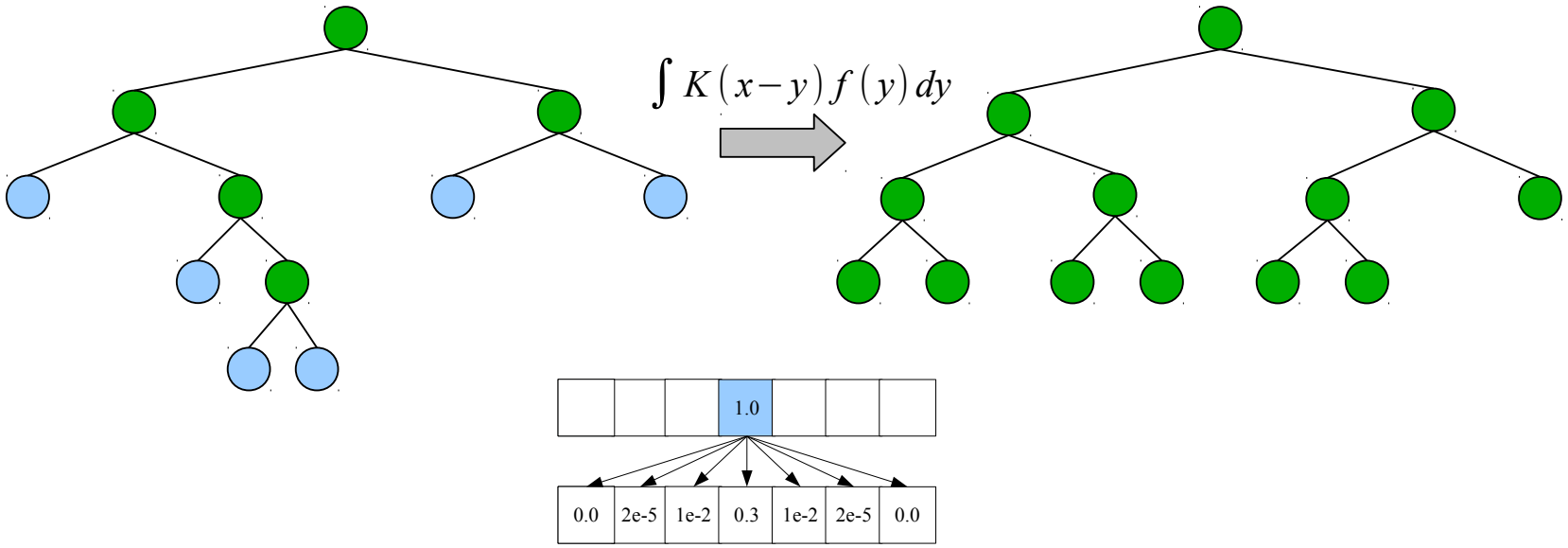

**Addition** is (most straightforwardly) performed in the compressed form. Coefficients are simply added with missing nodes being treated as if zero.




**Differentiation** (for simplicity here using central differences and Dirichlet boundary conditions) is applied in the scaling function basis. To compute the derivative of the function in the box corresponding to a leaf node, we require the coefficients from the neighboring boxes at the same level.

- If the neighboring leaf nodes exist, all is easy.
- If it exists at a higher level,we can make the coefficients by recurring down from the parent using the two-scale relation.
- If the neighbor exists at a finer scale, we must recur down until both neighbors are at the same level.

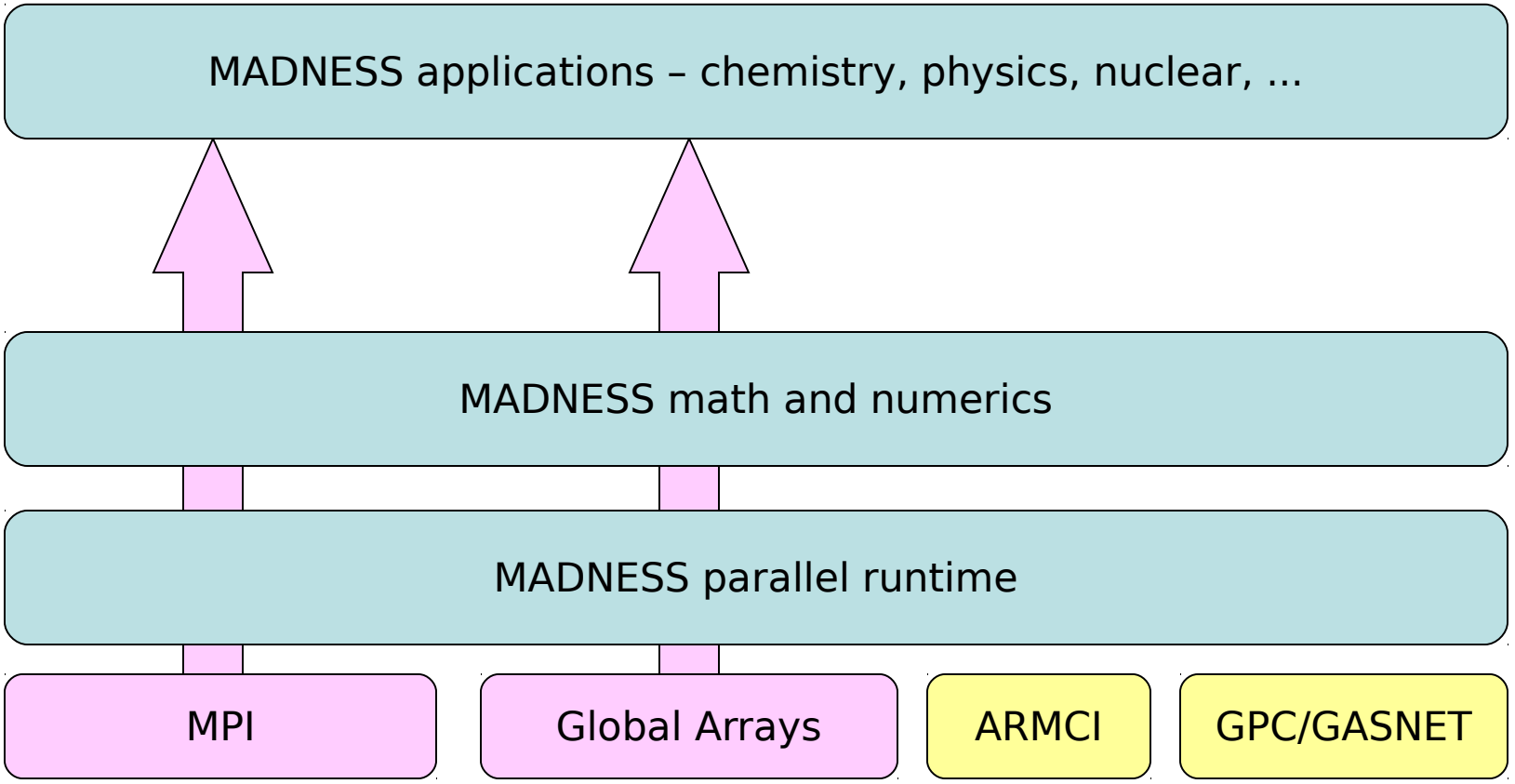
Hence, phrased as parallel computation on all leaf nodes , differentiation must search for neighbors in the tree at the same and higher levels, and may initiate computation at lower levels. It can also be phrased as a recursive descent of the tree, which can have advantages in reducing the amount of probes up the tree for parents of neighbors (esp. in higher dimensions).




**Convolution** The first step is to compress into non-standard form with scaling function and wavelet coefficients at each interior node. Then, we can independently compute the contribution of each box (node) to the result *at the same level of the tree*. Depending upon dimensionality, accuracy, and the kernel ($K$), we usually only need to compute the contributions of a box to itself and its immediate neighbors. The support (i.e., level of refinement) of the result is very dependent on the kernel. Here we consider convolution with a Gaussian (Green's function for the heat equation) which is a *smoothing* operator. After the computation is complete, we must sum down the tree to recover the standard form.

Hence, phrased as computation on all the nodes in non-standard form, convolution requires compression and reconstruction, and during the computation communicates across the tree at the same level to add results into neighboring boxes and up to connect new nodes to parents.

# MADNESS architecture




Intel Thread Building Blocks now the target for the intranode runtime
May more adopt more of TBB functionality
Open Community Runtime of great interest

# Runtime Objectives

- Scalability to 1+M processors ASAP
- Runtime responsible for
  - scheduling and placement,
  - managing dependencies & hiding latency
- Compatible with existing models (MPI, GA)
- Borrow successful concepts from Cilk, Charm++, Python, HPCS languages

# Why a new runtime?

- MADNESS computation is irregular & dynamic
  - 1000s of dynamically-refined meshes changing frequently & independently (to guarantee precision)

- Because we wanted to make MADNESS itself easier to write not just the applications using it
  - We explored implementations with MPI, Global Arrays, and Charm++ and all were inadequate

- MADNESS is helping drive
  - One-sided operations in MPI-3, DOE projects in fault tolerance, ...

# Key runtime elements

- Futures for hiding latency and automating dependency management
- Global names and name spaces
- Non-process centric computing
  - One-sided messaging between objects
  - Retain place=process for MPI/GA legacy compatibility
- Dynamic load balancing
  - Data redistribution, work stealing, randomization

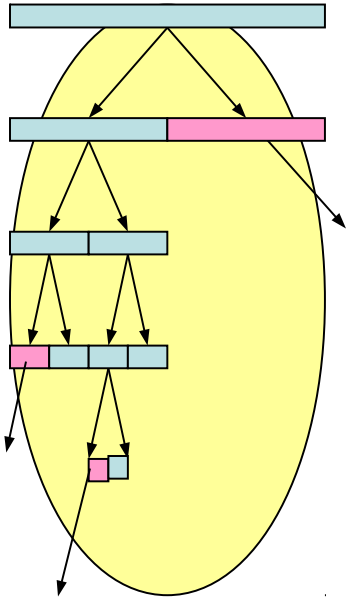
# Futures

- Result of an asynchronous computation
  - Cilk, Java, HPCLs, C++0x
- Hide latency due to communication or computation
- Management of dependencies
  - Via callbacks

```
int f(int arg);
ProcessId me, p;

Future<int> r0=task(p, f, 0);
Future<int> r1=task(me, f, r0);

// Work until need result

cout << r0 << r1 << endl;
```

Process “me” spawns a new task in process “p” to execute `f(0)` with the result eventually returned as the value of future `r0`. This is used as the argument of a second task whose execution is deferred until its argument is assigned. Tasks and futures can register multiple local or remote callbacks to express complex and dynamic dependencies.

# Virtualization of data and tasks



Future:
- MPI rank
- probe()
- set()
- get()

Task:
- Input parameters
- Output parameters
- probe()
- run()
- get()

```
Future Compress(tree):
      Future left = Compress(tree.left)
      Future right = Compress(tree.right)
      return Task(Op, left, right)

Compress(tree)
Wait for all tasks to complete
```

**Benefits: Communication latency & transfer time largely hidden**
**Much simpler composition than explicit message passing**
**Positions code to use "intelligent" runtimes with work stealing**
**Positions code for efficient use of multi-core chips**
**Locality-aware and/or graph-based scheduling**

# Global Names

- Objects with global names with different state in each process
  - C.f. shared[threads] in UPC; co-Array
- Non-collective constructor; deferred destructor
  - Eliminates synchronization

```
class A : public WorldObject<A>
{
  int f(int);
};
ProcessID p;
A a(world);
Future<int> b =
  a.task(p,&A::f,0);
```

A task is sent to the instance of a in process p. If this has not yet been constructed the message is stored in a pending queue. Destruction of a global object is deferred until the next user synchronization point.

```
#define WORLD_INSTANTIATE_STATIC_TEMPLATES
#include <world/world.h>
using namespace madness;
class Foo : public WorldObject<Foo> {
    const int bar;
public:
    Foo(World& world, int bar) : WorldObject<Foo>(world), bar(bar)
                {process_pending();}

    int get() const {return bar;}
};
int main(int argc, char** argv) {
    MPI::Init(argc, argv);
    madness::World world(MPI::COMM_WORLD);

    Foo a(world,world.rank()), b(world,world.rank()*10)

    for (ProcessID p=0; p<world.size(); p++) {
        Future<int> futa = a.send(p,&Foo::get);
        Future<int> futb = b.send(p,&Foo::get);
        // Could work here until the results are available
        MADNESS_ASSERT(futa.get() == p);
        MADNESS_ASSERT(futb.get() == p*10);
    }
    world.gop.fence();
    if (world.rank() == 0) print("OK!");
    MPI::Finalize();
}
```

*Figure 1: Simple client-server program implemented using WorldObject.*

```
#define WORLD_INSTANTIATE_STATIC_TEMPLATES
#include <world/world.h>

using namespace std;
using namespace madness;

class Array : public WorldObject<Array> {
   vector<double> v;
public:
   /// Make block distributed array with size elements
   Array(World& world, size_t size)
      : WorldObject<Array>(world), v((size-1)/world.size()+1)
   {
      process_pending();
   };

   /// Return the process in which element i resides
   ProcessID owner(size_t i) const {return i/v.size();};

   Future<double> read(size_t i) const {
      if (owner(i) == world.rank())
         return Future<double>(v[i-world.rank()*v.size()]);
      else
         return send(owner(i), &Array::read, i);
   };

   Void write(size_t i, double value) {
      if (owner(i) == world.rank())
         v[i-world.rank()*v.size()] = value;
      else
         send(owner(i), &Array::write, i, value);
      return None;
   };
};
```

```
int main(int argc, char** argv) {
   initialize(argc, argv);
   madness::World world(MPI::COMM_WORLD);

   Array a(world, 10000), b(world, 10000);

   // Without regard to locality, initialize a and b
   for (int i=world.rank(); i<10000; i+=world.size()) {
      a.write(i, 10.0*i);
      b.write(i,  7.0*i);
   }
   world.gop.fence();

   // All processes verify 100 random values from each array
   for (int j=0; j<100; j++) {
      size_t i = world.rand()%10000;
      Future<double> vala = a.read(i);
      Future<double> valb = b.read(i);
      // Could do work here until results are available
      MADNESS_ASSERT(vala.get() == 10.0*i);
      MADNESS_ASSERT(valb.get() ==  7.0*i);
   }
   world.gop.fence();

   if (world.rank() == 0) print("OK!");
   finalize();
}
```

*Complete example program illustrating the implementation and use of a crude, block-distributed array upon the functionality of* `WorldObject`.

# Global Namespaces

- Specialize global names to containers
  - Hash table, arrays, ...
- Replace global pointer (process+local pointer) with more powerful concept
- User definable map from keys to “owner” process

```
class Index;  // Hashable
class Value {
  double f(int);
};

WorldContainer<Index,Value> c;
Index i,j;  Value v;
c.insert(i,v);
Future<double> r =
  c.task(j,&Value::f,666);
```

A container is created mapping indices to values.

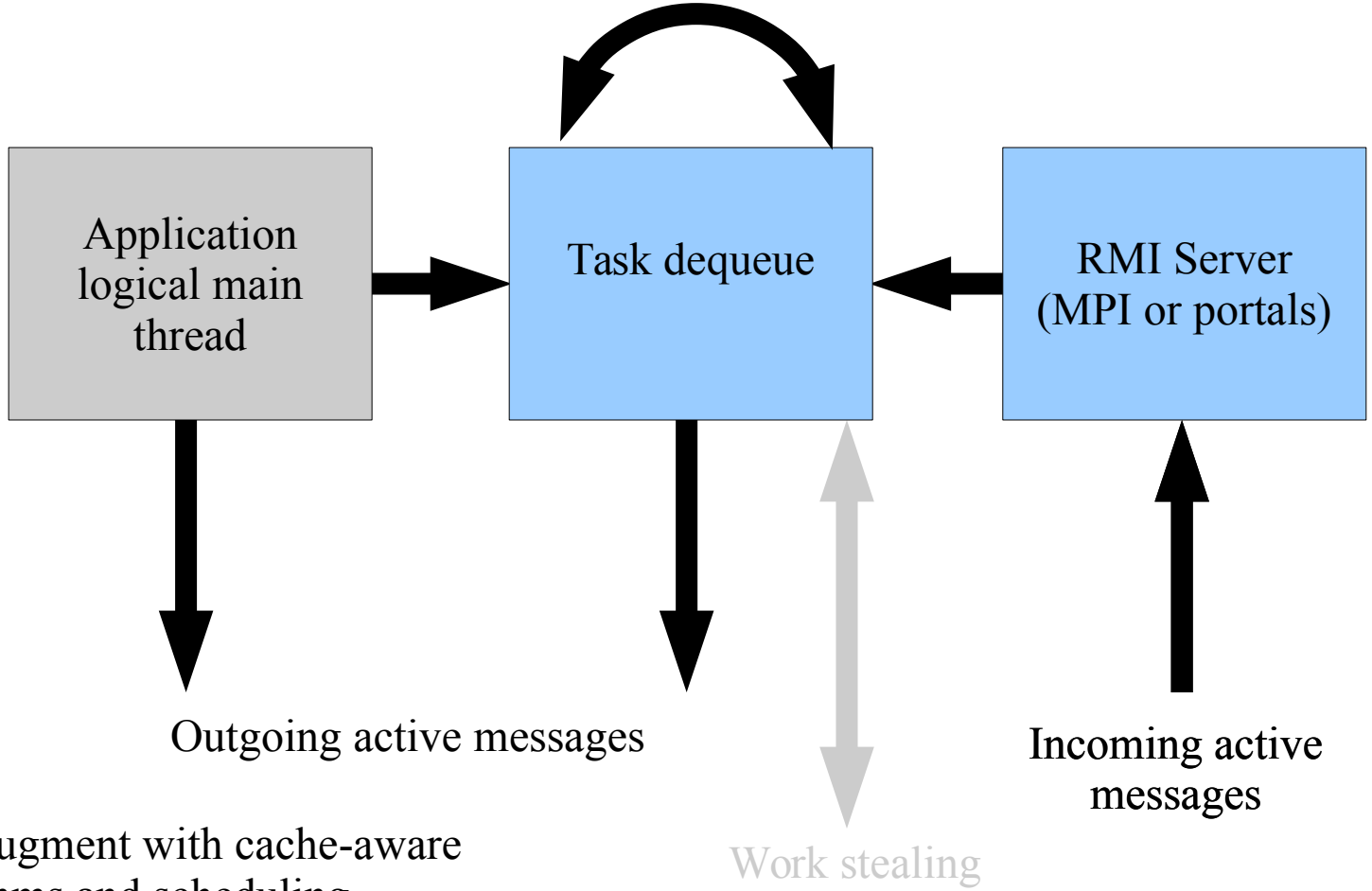
A value is inserted into the container.

A task is spawned in the process owning key `j` to invoke `c[j].f(666)`.



Namespaces are a large part of the elegance of Python and success of Charm++ (chares+arrays)

# Multi-threaded architecture

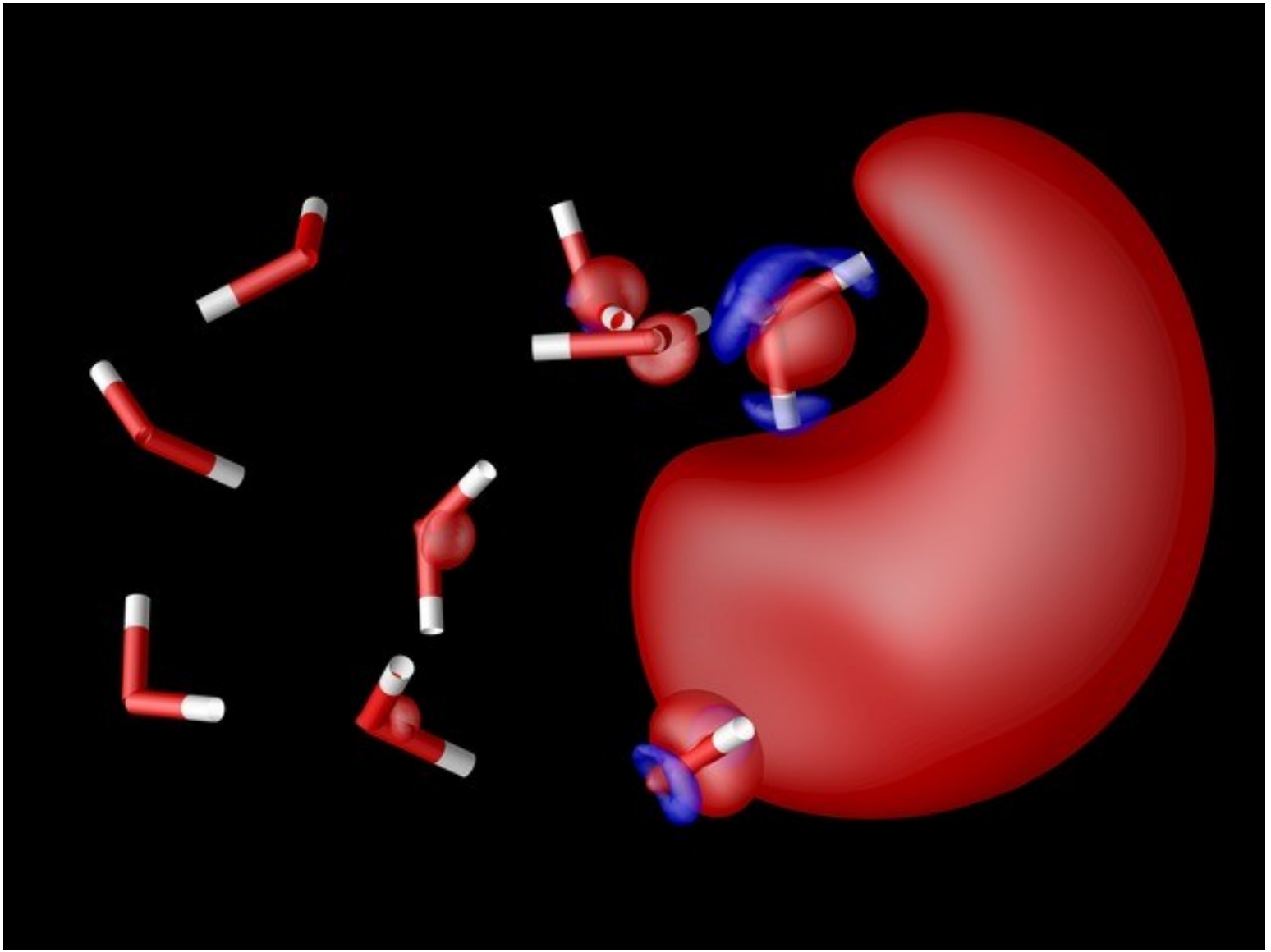



Must augment with cache-aware algorithms and scheduling

# Some issues

- Excessive global barriers
  - Termination detection for global algorithms on distributed mutable data structures
- Messy, nearly redundant code expressing variants of algorithms on multiple trees
  - Need some templates / code generation
- Need efficient and easy way to aggregate data/work to exploit GPGPUs
- Efficient kernels for GPGPUs (single SM)
  - Non-square matrices, shortish loops – performance problem
- Switching between single-/multi-thread tasks
- Efficient multi-threaded code for thread units sharing L1 (e.g., BGQ, Xeon Phi)
- Multiple interoperable DSLs embedded in or generating general purpose language
- Kitchen sink environment – full interoperability between runtimes, data structures, external I/O libraries, etc.

# Molecular Electronic Structure



Energy and gradients

ECPs coming (Sekino, Thornton)

Response properties (Vasquez, Yokoi, Sekino)

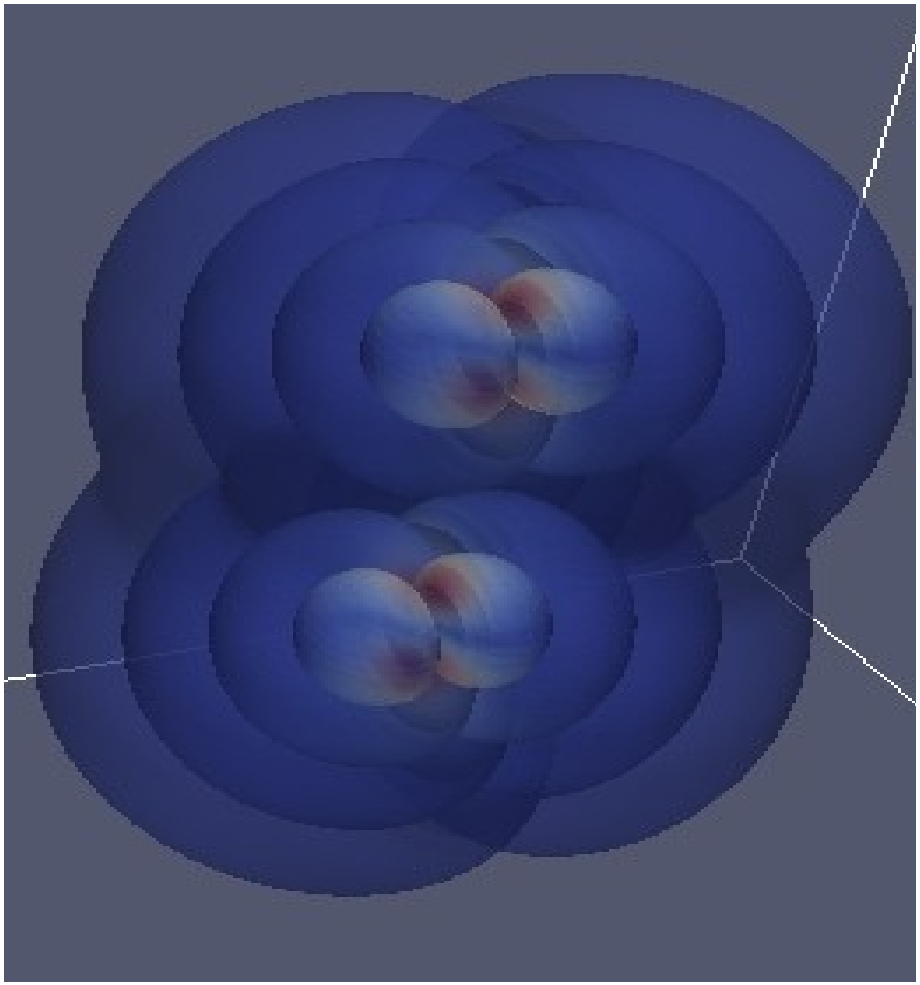
Still not as functional as previous Python version of Yanai

*Spin density of solvated electron*

# Nuclear physics

J. Pei, G.I. Fann, Y. Ou,
W. Nazarewicz
UT/ORNL

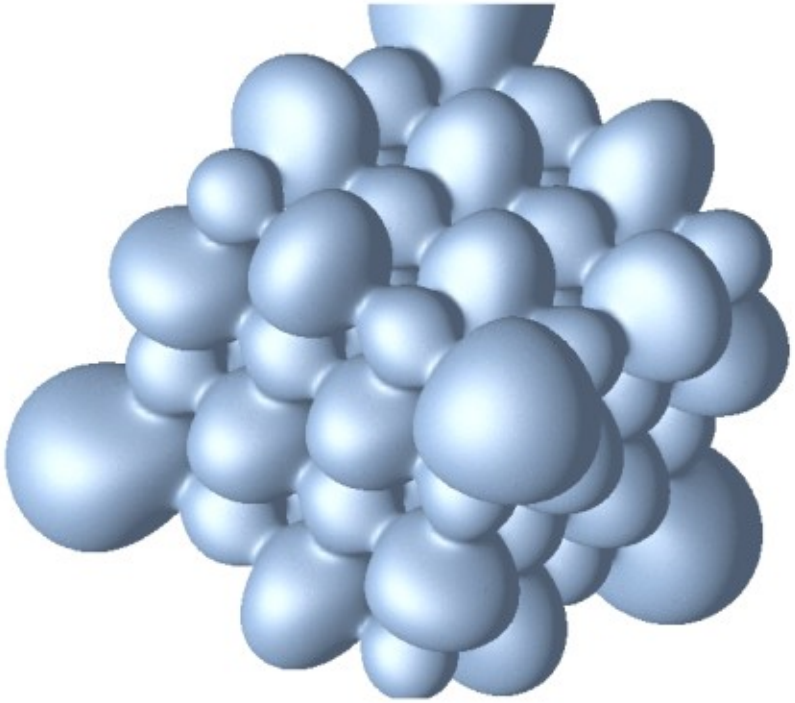
- DOE UNDEF
- Nuclei & neutron matter
- ASLDA
- Hartree-Fock Bogliobulov
- Spinors
- Gamov states



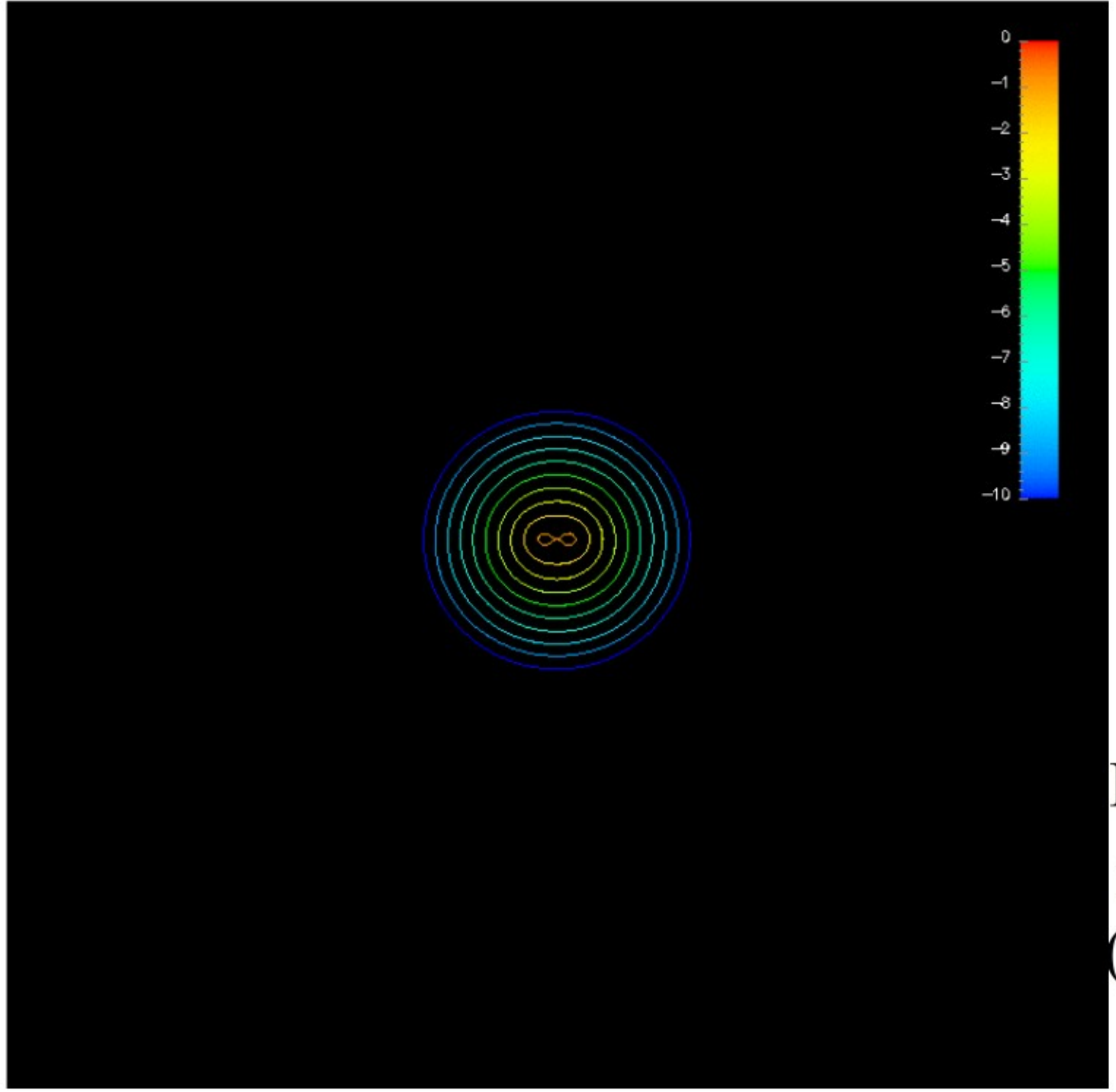
Imaginary part of the seventh eigen function
two-well Wood-Saxon potential

# Solid-state electronic structure

- Thornton, Eguiluz and Harrison (UT/ORNL)
  - NSF OCI-0904972: Computational chemistry and physics beyond the petascale
- Full band structure with LDA and HF for periodic systems
- In development: hybrid functionals, response theory, post-DFT methods such as GW and model many-body Hamiltonians via Wannier functions



Coulomb potential isosurface in LiF



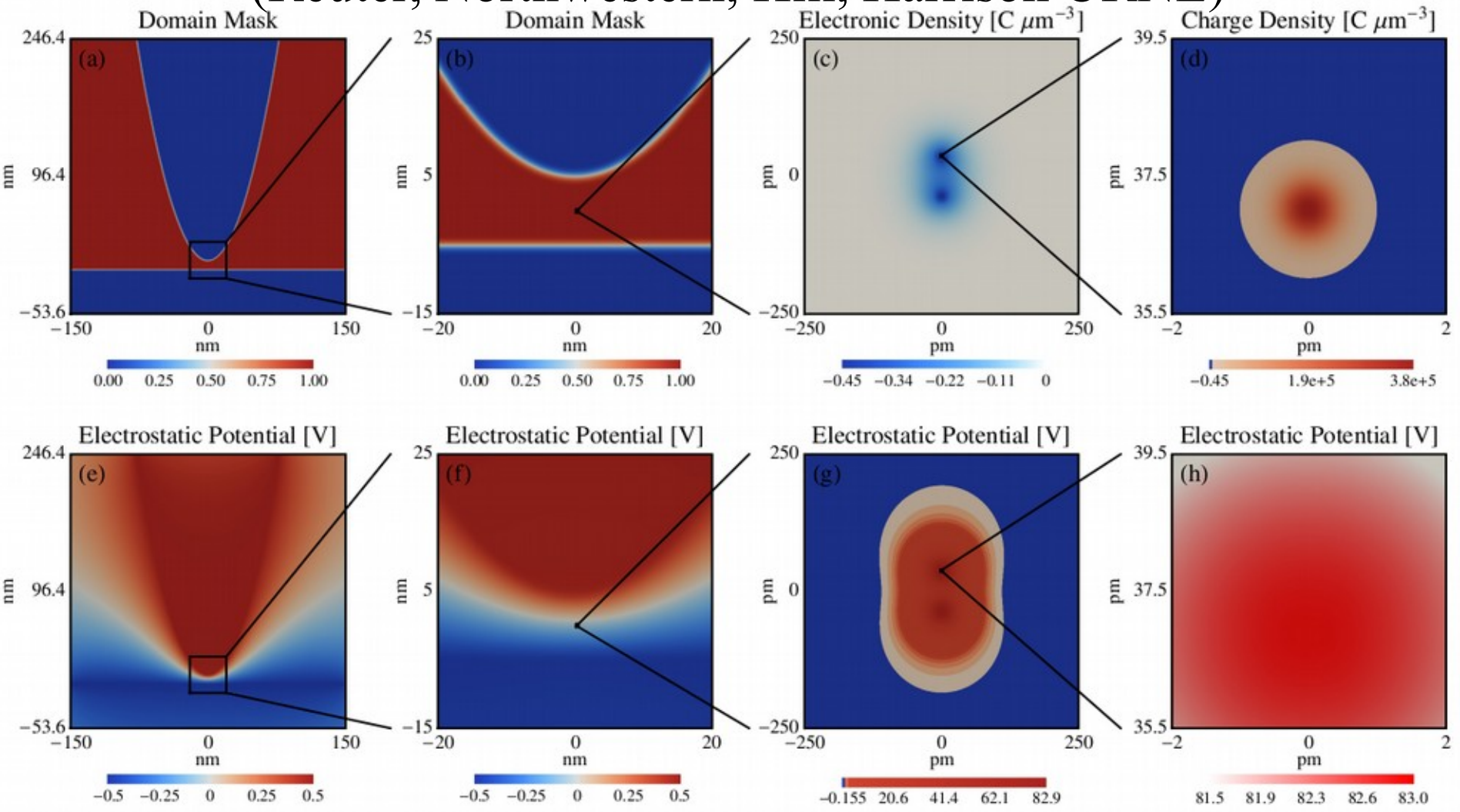

Time dependent electronic structure

Vence, Krstic, Harrison UT/ORNL

$H_2^+$ molecule in laser field (fixed nuclei)

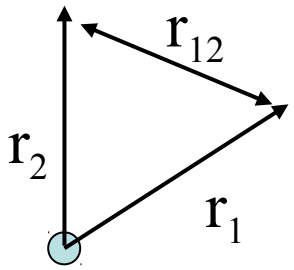
# Nanoscale photonics

## (Reuter, Northwestern; Hill, Harrison ORNL)




Diffuse domain approximation for interior boundary value problem; long-wavelength Maxwell equations; Poisson equation; Micron-scale Au tip 2 nm above Si surface with H2 molecule in gap – $10^7$ difference between shortest and longest length scales.

# Electron correlation (6D)




- All defects in mean-field model are ascribed to electron correlation
- Singularities in Hamiltonian imply for a two-electron atom

$$\Psi(r_1, r_2, r_{12}) = 1 + \frac{1}{2} r_{12} + \cdots \quad \text{as} \quad r_{12} \to 0$$

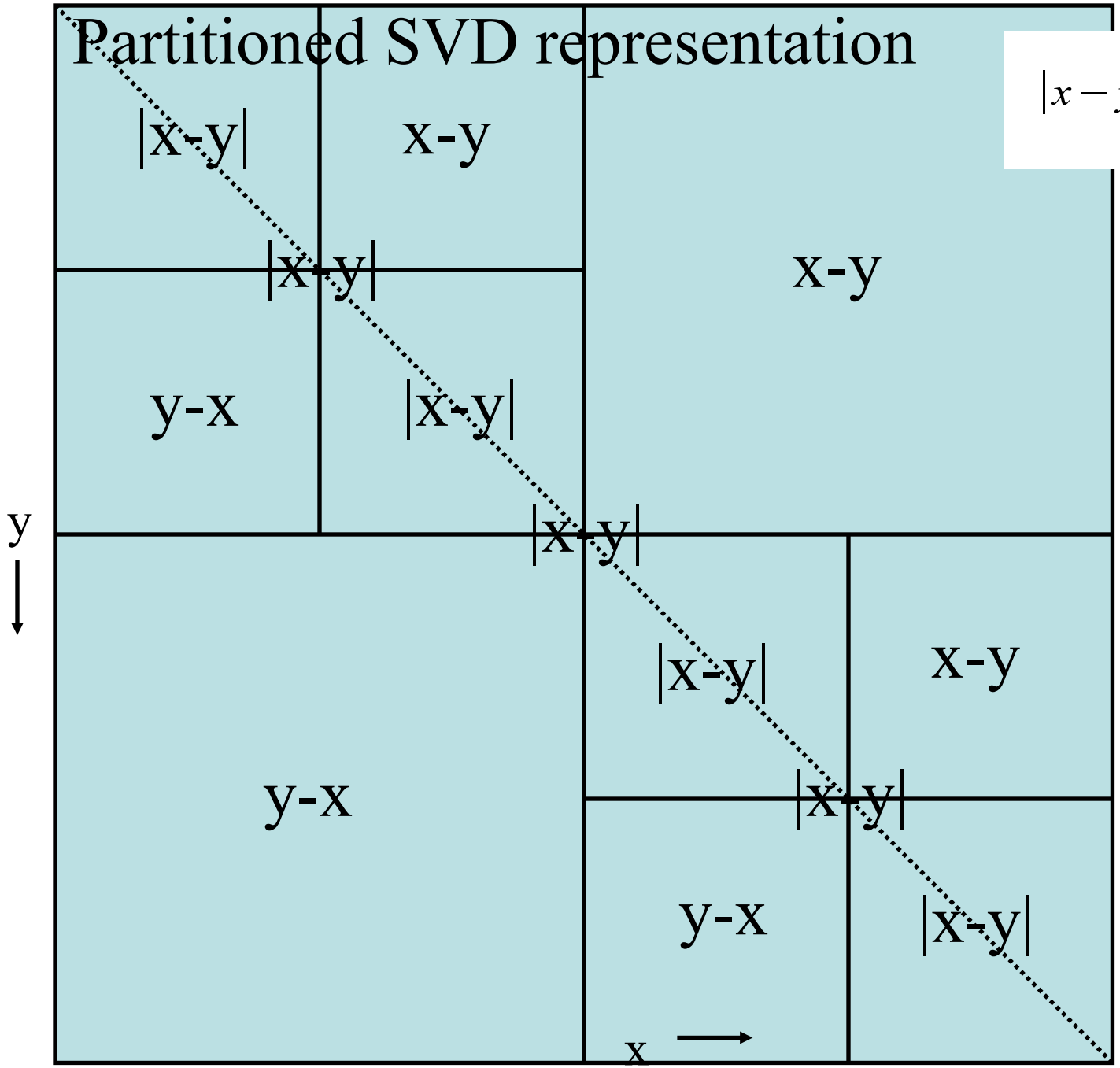
- Include the inter-electron distance in the wavefunction
  - E.g., Hylleraas 1938 wavefunction for He

$$\Psi(r_1, r_2, r_{12}) = \exp(-\xi(r_1 + r_2))(1 + a\, r_{12} + \cdots)$$

  - Potentially very accurate, but not systematically improvable, and (until recently) not computationally feasible for many-electron systems
- Configuration interaction expansion – slowly convergent

$$\Psi(r_1, r_2, \ldots) = \sum_i c_i \left| \Phi_1^{(i)}(r_1) \Phi_2^{(i)}(r_2) \ldots \right|$$

# Partitioned SVD representation




$$|x-y| = \sum_{\mu=1}^{r} f_\mu(x)\, g_\mu(y)$$

$r$ = separation rank

In 3D, ideally must be one box removed from the diagonal

Diagonal box has full rank

Boxes touching diagonal (face, edge, or corner) have increasingly low rank

Away from diagonal $r = O(-\log \varepsilon)$

# The way forward demands a change in paradigm

- by us chemists, the funding agencies, and the supercomputer centers

- A communal effort recognizing the increased cost and complexity of code development for modern theory beyond the petascale
- Coordination between agencies to develop and deploy new simulation capabilities in sustainable manner
- Re-emphasizing basic and advanced theory and computational skills in undergraduate and graduate education

S I C M $^2$

Sustainable Software for Chemistry and Materials



# A Sustainable Software Innovation Institute for Computational Chemistry and Materials Modeling (S2I2C2M2)

## Principal Investigator

T. Daniel Crawford (Virginia Tech)

## Co-Principal Investigators

Robert J. Harrison (Stony Brook U.)

Anna Krylov (U. Southern California)

Theresa Windus (Iowa State U.)

## Senior Personnel

Emily Carter (Princeton U.)
Erik Deumens (U. Florida)
Martin Head-Gordon (U. C. Berkeley)
David McDowell (Georgia Tech)
Manish Parashar (Rutgers U.)
Beverly Sanders (U. Florida)
David Sherrill (Georgia Tech)
Masha Sosonkina (Iowa State U.)

Edmund Chow (Georgia Tech)
Mark Gordon (Iowa State U.)
Todd Martinez (Stanford U.)
Vijay Pande (Stanford U.)
Ram Ramanujam (LSU)
Bernhard Schlegel (Wayne State U.)
Lyudmila Slipchenko (Purdue U.)
Edward Valeev (Virginia Tech)

Ross Walker (San Diego Supercomputing Center)

## NSF SI$^2$ and Other Collaborators

Jochen Autschbach (U. Buffalo)
John F. Stanton (Senior Kibbitzer) (U. Texas)
Garnet Chan (Princeton U.)
So Hirata (U. Illinois)
Toru Shiozaki (Northwestern U.)

**http://s2i2.org**

# Summary

- We need radical changes in how we compose scientific S/W
  - Complexity at limits of cost and human ability
  - Need extensible tools/languages with support for code transformation not just translation
- Students need to be prepared for computing and data in 2020+ not as it was in 2000 and before
  - Pervasive, massive parallelism
  - Bandwidth limited computation and analysis
- An intrinsically multidisciplinary activity

# Funding


- DOE: Exascale co-design, SciDAC, Office of Science divisions of Advanced Scientific Computing Research and Basic Energy Science, under contract DE-AC05-00OR22725 with Oak Ridge National Laboratory, in part using the National Center for Computational Sciences.
- DARPA HPCS2: HPCS programming language evaluation
- NSF CHE-0625598: Cyber-infrastructure and Research Facilities: Chemical Computations on Future High-end Computers
- NSF CNS-0509410: CAS-AES: An integrated framework for compile-time/run-time support for multi-scale applications on high-end systems
- NSF OCI-0904972:  Computational Chemistry and Physics Beyond the Petascale

**Do new science with**

**O(1) programmers**
**O(100,000) nodes**
**O(100,000,000) cores**
**O(1,000,000,000) threads & growing**

- Increasing intrinsic complexity of science
- Complexity kills … sequential or parallel
  - Expressing concurrency at extreme scale
  - Managing the memory hierarchy
- Semantic gap (Colella)
  - Why are equations O(100) lines but program is O(1M)
  - What's in the semantic gap – and how to shrink it?

# Wish list

- Eliminate gulf between theoretical innovation in small groups and realization on high-end computers
- Eliminate the semantic gap so that efficient parallel code is no harder than doing the math
- Enable performance-portable “code” that can be automatically migrated to future architectures
- Reduce cost at all points in the life cycle

- Much of this is pipe dream – but what can we aspire to?

# Scientific vs. WWW or mobile software


- Why are we not experiencing similar exponential growth in functionality?
  - Level of investment; no. of developers?
  - Lack of software interoperability and standards?
  - Competition not cooperation between groups?
  - Shifting scientific objectives?
  - Are our problems intrinsically harder?
  - Failure to embrace/develop higher levels of composition?
  - Different hardware complexity?

```
107              if(ierr.ne.0)stop 'DEALLOC 02'
108              call wallmark
109              call make_ghost
110              call io_result(5)
111            end if
112          end if
113
114          err     = 0.d0
115          rsdl    = 0.d0
116          iterate = 0
117  !$omp parallel do
118  !$omp& reduction(+:err)
119  !$omp& reduction(+:rsdl)
120  !$omp& reduction(+:iterate)
121  !$omp& private(icube0)
122  !$omp& private(jmin,jmax,lmin,lmax)
123  !$omp& private(j,l)
124  !$omp& private(u0,u1,u2,u3,u4,u5,u6,u7,u8)
125  !$omp& private(w0,w1,w2,w3,w4,w5,w6,w7,w8)
126  !$omp& private(ux,uz,wx,wz,uxx,wzz,uxz,wzx,cnt,adv)
127  !$omp& private(ds,rs,p0,p1,p2,p3,p4,p_a,err0)
128  !$omp& private(iter1)
129          do icube0 = 1, n_cube
130
```

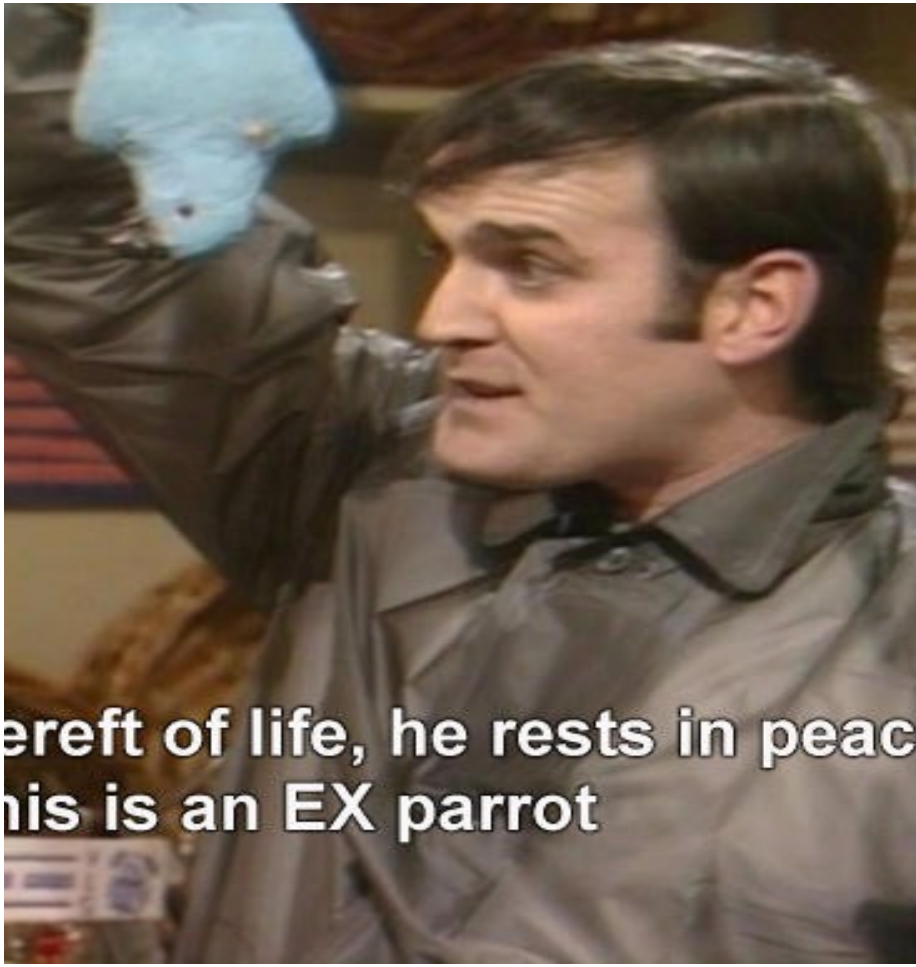
# How do we write code for a machine that does not yet exist?

- Nothing too exotic, e.g., the mix of SIMD and scalar units, registers, massive multi-threading, software/hardware managed cache, fast/slow & local/remote memory that we expect in 2018+
- Answer 1: presently cannot
  - but it's imperative that we learn how and deploy the necessary tools
- Answer 2: don't even try!
  - where possible generate code from high level specs
  - provides tremendous agility and freedom to explore diverse architectures

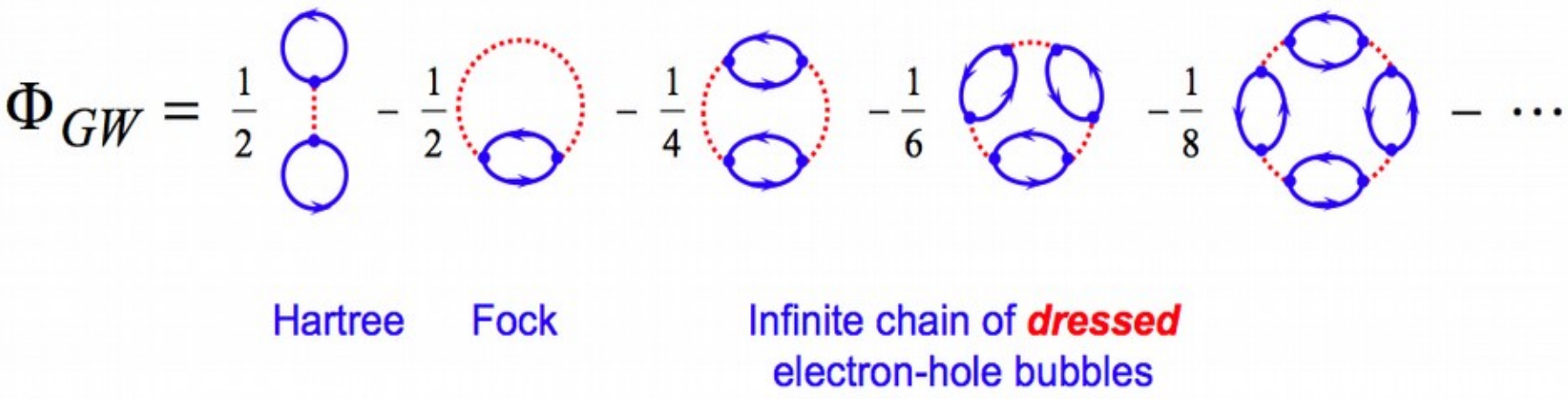
# Dead code

7 December 1969

- Requires human labor
  - to migrate to future architectures, or
  - to exploit additional concurrency, or
  - ...
- By these criteria most extant code is dead
- Sanity check
  - How much effort is required to port to hybrid cpu+GPGPU?



# The language of many-body physics



# CCSD Doubles Equation

```
hbar[a,b,i,j] == sum[f[b,c]*t[i,j,a,c],{c}] -sum[f[k,c]*t[k,b]*t[i,j,a,c],{k,c}] +sum[f[a,c]*t[i,j,c,b],{c}] -sum[f[k,c]*t[k,a]*t[i,j,c,b],{k,c}]
    -sum[f[k,j]*t[i,k,a,b],{k}] -sum[f[k,c]*t[j,c]*t[i,k,a,b],{k,c}] -sum[f[k,i]*t[j,k,b,a],{k}] -sum[f[k,c]*t[i,c]*t[j,k,b,a],{k,c}]
    +sum[t[i,c]*t[j,d]*v[a,b,c,d],{c,d}] +sum[t[i,j,c,d]*v[a,b,c,d],{c,d}] +sum[t[j,c]*v[a,b,i,c],{c}] -sum[t[k,b]*v[a,k,i,j],{k}]
    +sum[t[i,c]*v[b,a,j,c],{c}] -sum[t[k,a]*v[b,k,j,i],{k}] -sum[t[k,d]*t[i,j,c,b]*v[k,a,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,b,d]*v[k,a,c,d],
    {k,c,d}] -sum[t[j,c]*t[k,b]*v[k,a,c,i],{k,c}] +2*sum[t[j,k,b,c]*v[k,a,c,i],{k,c}] -sum[t[j,k,c,b]*v[k,a,c,i],{k,c}]
    -sum[t[i,c]*t[j,d]*t[k,b]*v[k,a,d,c],{k,c,d}] +2*sum[t[k,d]*t[i,j,c,b]*v[k,a,d,c],{k,c,d}] -sum[t[k,b]*t[i,j,c,d]*v[k,a,d,c],{k,c,d}]
    -sum[t[j,d]*t[i,k,c,b]*v[k,a,d,c],{k,c,d}] +2*sum[t[i,c]*t[j,k,b,d]*v[k,a,d,c],{k,c,d}] -sum[t[i,c]*t[j,k,d,b]*v[k,a,d,c],{k,c,d}]
    -sum[t[j,k,b,c]*v[k,a,i,c],{k,c}] -sum[t[i,c]*t[k,b]*v[k,a,j,c],{k,c}] -sum[t[i,k,c,b]*v[k,a,j,c],{k,c}]
    -sum[t[i,c]*t[j,d]*t[k,a]*v[k,b,c,d],{k,c,d}] -sum[t[k,d]*t[i,j,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[k,a]*t[i,j,c,d]*v[k,b,c,d],{k,c,d}]
    +2*sum[t[j,d]*t[i,k,a,c]*v[k,b,c,d],{k,c,d}] -sum[t[j,d]*t[i,k,c,a]*v[k,b,c,d],{k,c,d}] -sum[t[i,c]*t[j,k,d,a]*v[k,b,c,d],{k,c,d}]
    -sum[t[i,c]*t[k,a]*v[k,b,c,j],{k,c}] +2*sum[t[i,k,a,c]*v[k,b,c,j],{k,c}] -sum[t[i,k,c,a]*v[k,b,c,j],{k,c}]
    +2*sum[t[k,d]*t[i,j,a,c]*v[k,b,d,c],{k,c,d}] -sum[t[j,d]*t[i,k,a,c]*v[k,b,d,c],{k,c,d}] -sum[t[j,c]*t[k,a]*v[k,b,i,c],{k,c}]
    -sum[t[j,k,c,a]*v[k,b,i,c],{k,c}] -sum[t[i,k,a,c]*v[k,b,j,c],{k,c}] +sum[t[i,c]*t[j,d]*t[k,a]*t[l,b]*v[k,l,c,d],{k,l,c,d}]
    -2*sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[k,l,c,d],{k,l,c,d}]
    +sum[t[k,a]*t[l,b]*t[i,j,c,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,c,d],{k,l,c,d}]
    -2*sum[t[j,d]*t[l,b]*t[i,k,a,c]*v[k,l,c,d],{k,l,c,d}] +sum[t[j,d]*t[l,b]*t[i,k,c,a]*v[k,l,c,d],{k,l,c,d}]
    -2*sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[k,l,c,d],{k,l,c,d}]
    +sum[t[i,c]*t[l,b]*t[j,k,d,a]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,c,d],{k,l,c,d}]
    +4*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,c,d],{k,l,c,d}]
    -2*sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,d,b]*v[k,l,c,d],
    {k,l,c,d}] +sum[t[i,c]*t[j,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}] +sum[t[i,j,c,d]*t[k,l,a,b]*v[k,l,c,d],{k,l,c,d}]
    -2*sum[t[i,j,c,b]*t[k,l,a,d]*v[k,l,c,d],{k,l,c,d}] -2*sum[t[i,j,a,c]*t[k,l,b,d]*v[k,l,c,d],{k,l,c,d}] +sum[t[j,c]*t[k,b]*t[l,a]*v[k,l,c,i],
    {k,l,c}] +sum[t[l,c]*t[j,k,b,a]*v[k,l,c,i],{k,l,c}] -2*sum[t[l,a]*t[j,k,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[l,a]*t[j,k,c,b]*v[k,l,c,i],{k,l,c}]
    -2*sum[t[k,c]*t[j,l,b,a]*v[k,l,c,i],{k,l,c}] +sum[t[k,a]*t[j,l,b,c]*v[k,l,c,i],{k,l,c}] +sum[t[k,b]*t[j,l,c,a]*v[k,l,c,i],{k,l,c}]
    +sum[t[j,c]*t[l,k,a,b]*v[k,l,c,i],{k,l,c}] +sum[t[i,c]*t[k,a]*t[l,b]*v[k,l,c,j],{k,l,c}] +sum[t[l,c]*t[i,k,a,b]*v[k,l,c,j],{k,l,c}]
    -2*sum[t[l,b]*t[i,k,a,c]*v[k,l,c,j],{k,l,c}] +sum[t[l,b]*t[i,k,c,a]*v[k,l,c,j],{k,l,c}] +sum[t[i,c]*t[k,l,a,b]*v[k,l,c,j],{k,l,c}]
    +sum[t[j,c]*t[l,d]*t[i,k,a,b]*v[k,l,d,c],{k,l,c,d}] +sum[t[j,d]*t[l,b]*t[i,k,a,c]*v[k,l,d,c],{k,l,c,d}]
    +sum[t[j,d]*t[l,a]*t[i,k,c,b]*v[k,l,d,c],{k,l,c,d}] -2*sum[t[i,k,c,d]*t[j,l,b,a]*v[k,l,d,c],{k,l,c,d}]
    -2*sum[t[i,k,a,c]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,c,a]*t[j,l,b,d]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,b]*t[j,l,c,d]*v[k,l,d,c],
    {k,l,c,d}] +sum[t[i,k,c,b]*t[j,l,d,a]*v[k,l,d,c],{k,l,c,d}] +sum[t[i,k,a,c]*t[j,l,d,b]*v[k,l,d,c],{k,l,c,d}] +sum[t[k,a]*t[l,b]*v[k,l,i,j],
    {k,l}] +sum[t[k,l,a,b]*v[k,l,i,j],{k,l}] +sum[t[k,b]*t[l,d]*t[i,j,a,c]*v[l,k,c,d],{k,l,c,d}] +sum[t[k,a]*t[l,d]*t[i,j,c,b]*v[l,k,c,d],
    {k,l,c,d}] +sum[t[i,c]*t[l,d]*t[j,k,b,a]*v[l,k,c,d],{k,l,c,d}] -2*sum[t[i,c]*t[l,a]*t[j,k,b,d]*v[l,k,c,d],{k,l,c,d}]
    +sum[t[i,c]*t[l,a]*t[j,k,d,b]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,j,c,b]*t[k,l,a,d]*v[l,k,c,d],{k,l,c,d}] +sum[t[i,j,a,c]*t[k,l,b,d]*v[l,k,c,d],
    {k,l,c,d}] -2*sum[t[l,c]*t[i,k,a,b]*v[l,k,c,j],{k,l,c}] +sum[t[l,b]*t[i,k,a,c]*v[l,k,c,j],{k,l,c}] +sum[t[l,a]*t[i,k,c,b]*v[l,k,c,j],{k,l,c}]
    +v[a,b,i,j]
```

$$\bar{h}_{ij}^{ab} = \left\langle \begin{matrix} a\,b \\ i\,j \end{matrix} \middle| e^{-\hat{T}_1-\hat{T}_2}\,\hat{H}\,e^{\hat{T}_1+\hat{T}_2} \middle| 0 \right\rangle$$

# The Tensor Contraction Engine: A Tool for Quantum Chemistry


**Oak Ridge National Laboratory**
***David E. Bernholdt***, Venkatesh Choppella, *Robert Harrison*

**Pacific Northwest National Laboratory**
***So Hirata***

**Louisiana State University**
***J Ramanujam,***

**Ohio State University**
***Gerald Baumgartner,*** Alina Bibireata, Daniel Cociorva, Xiaoyang Gao, Sriram Krishnamoorthy, Sandhya Krishnan, Chi-Chung Lam, Quingda Lu, ***Russell M. Pitzer, P Sadayappan***, Alexander Sibiryakov

**University of Waterloo**
***Marcel Nooijen***, Alexander Auer


http://www.cis.ohio-state.edu/~gb/TCE/


Research at ORNL supported by the Laboratory Directed Research and Development Program. Research at PNNL supported by the Office of Basic Energy Sciences, U. S. Dept. of Energy. Research at OSU, Waterloo, and LSU supported by the National Science Foundation Information Technology Research Program

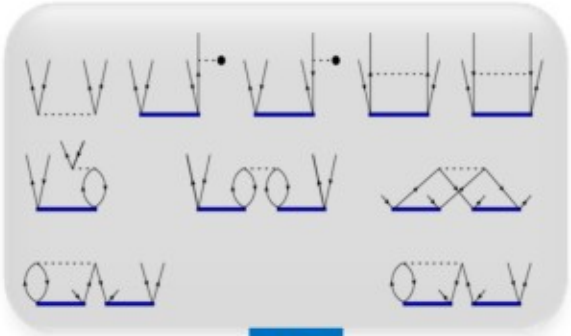
# Tensor Contraction Engine (TCE) (Kowalski, PNNL)

EMSL

Highly parallel codes are needed in order to apply the CC theories to larger molecular systems

Symbolic algebra systems for coding complicated tensor expressions: Tensor Contraction Engine (TCE)

| | | Expression[a] |
|---|---|---|
| $D_i^a t_i^a$ | = | $f_i^a + t_f^a I_f^a - t_n^a I_i'^n + t_{ni}^{fa} I_f^n + t_n^f v_{fi}^{na} - \frac{1}{2} t_{no}^{fa} v_{fi}^{no} + \frac{1}{2} t_{ni}^{fg} v_{fg}^{na} + \frac{1}{4} t_{ino}^{afg} v_{fg}^{no}$ |
| $D_{ij}^{ab} t_{ij}^{ab}$ | = | $v_{ij}^{ab} + P(a/b) I_f^a t_{ij}^{fb} - P(i/j) I_i^n t_{nj}^{ab} + \frac{1}{2} t_{ij}^{fg} I_{fg}'^{ab} + \frac{1}{2} t_{no}^{ab} I_{ij}^{no}$ $+ P(a/b)P(i/j) t_{in}^{af} I_{fj}^{nb} \frac{1}{2} P(a/b) I_{fg}^{na} t_{nij}^{fgb}$ $- \frac{1}{2} P(i/j) I_{fi}^{no} t_{noj}^{fab} + t_{nij}^{fab} I_f^n + P(i/j) t_i^f I_{fj}'^{ab} - P(a/b) t_n^a I_{ij}'^{nb} + \frac{1}{4} t_{ijno}^{abfg} v_{fg}^{no}$ |
| $D_{ijk}^{abc} t_{ijk}^{abc}$ | = | $P(a/bc) I_f^a t_{ijk}^{fbc} - P(i/jk) I_i^n t_{njk}^{abc} + \frac{1}{2} P(a/bc) t_{ijk}^{afg} I_{fg}^{bc} + \frac{1}{2} P(i/jk) t_{ino}^{abc} I_{jk}^{no}$ $+ P(ab/c) P(ij/k) t_{ijn}^{abf} I_{fk}^{nc} + P(a/bc) P(ij/k) t_{ij}^{af} I_{fk}^{bc} - P(ab/c) P(i/jk) t_{in}^{ab} I_{jk}^{nc}$ $+ t_{nijk}^{fabc} I_f^n + \frac{1}{2} P(a/bc) I_{fg}^{na} t_{nijk}^{fgbc} - P(i/jk) I_{fi}^{no} t_{nojk}^{fabc} + \frac{1}{4} t_{ijkno}^{abcfg} v_{fg}^{no}$ |
| $D_{ijkl}^{abcd} t_{ijkl}^{abcd}$ | = | $P(a/bcd) I_f^a t_{ijkl}^{fbcd} - P(i/jkl) I_i^n t_{njkl}^{abcd} + \frac{1}{2} P(ab/cd) t_{ijkl}^{abfg} I_{fg}^{cd} + \frac{1}{2} P(ij/kl) t_{ijno}^{abcd} I_{kl}^{no}$ $+ P(abc/d) P(ijk/l) t_{ijkn}^{abcf} I_{fl}^{nd} + P(ab/cd) P(ijk/l) t_{ijk}^{abf} I_{fl}^{cd} - P(abc/d) P(ij/kl) t_{ijn}^{abc} I_{kl}^{nd}$ $+ P(a/bcd) P(ij/kl) t_{ij}^{af} I_{fkl}'^{bcd} - P(ab/cd) P(i/jkl) t_{in}^{ab} I_{jkl}'^{ncd} + P(ab/cd) P(ij/kl) t_{ijn}^{abf} I_{jk}'^{ncd}$ $+ \frac{1}{2} P(abc/d) P(i/jkl) t_{ino}^{abc} I_{jkl}'^{nod} + t_{nijkl}^{fabcd} I_f^n + \frac{1}{2} P(a/bcd) I_{fg}^{na} t_{nijkl}^{fgbcd}$ $- \frac{1}{2} P(i/jkl) I_{fi}^{no} t_{nojkl}^{fabcd}$ |
| $D_{ijklm}^{abcde} t_{ijklm}^{abcde}$ | = | $P(a/bcde) I_f^a t_{ijklm}^{fbcde} - P(i/jklm) I_i^n t_{njklm}^{abcde} + \frac{1}{2} P(abc/de) t_{ijklm}^{abcfg} I_{fg}^{de} + \frac{1}{2} P(ijk/lm) t_{ijkno}^{abcde} I_{lm}^{no}$ $+ P(abcd/e) P(ijkl/m) t_{ijkln}^{abcdf} I_{fm}^{ne} + P(abc/de) P(ijk/lm) t_{ijkn}^{abcf} I_{flm}^{nde}$ $+ \frac{1}{2} P(abcd/e) P(ijk/lm) t_{ijno}^{abcd} I_{klm}'^{noe} + \frac{1}{2} P(ab/cde) P(ijk/lm) t_{ijkl}^{abfg} I_{fgm}'^{cde}$ $+ P(abc/de) P(ijk/lm) t_{ijkl}^{abcf} I_{fm}^{de} - P(abcd/e) P(ijk/lm) t_{ijkn}^{abcd} I_{lm}^{ne}$ $+ P(ab/cde) P(ijk/lm) t_{ijk}^{abf} I_{flm}'^{cde} - P(abc/de) P(ij/klm) t_{ijn}^{abc} I_{klm}^{nde}$ $+ P(a/bcde) P(ij/klm) t_{ij}^{af} I_{fklm}'^{bcde} - P(ab/cde) P(i/jklm) t_{in}^{ab} I_{jklm}'^{ncde}$ |



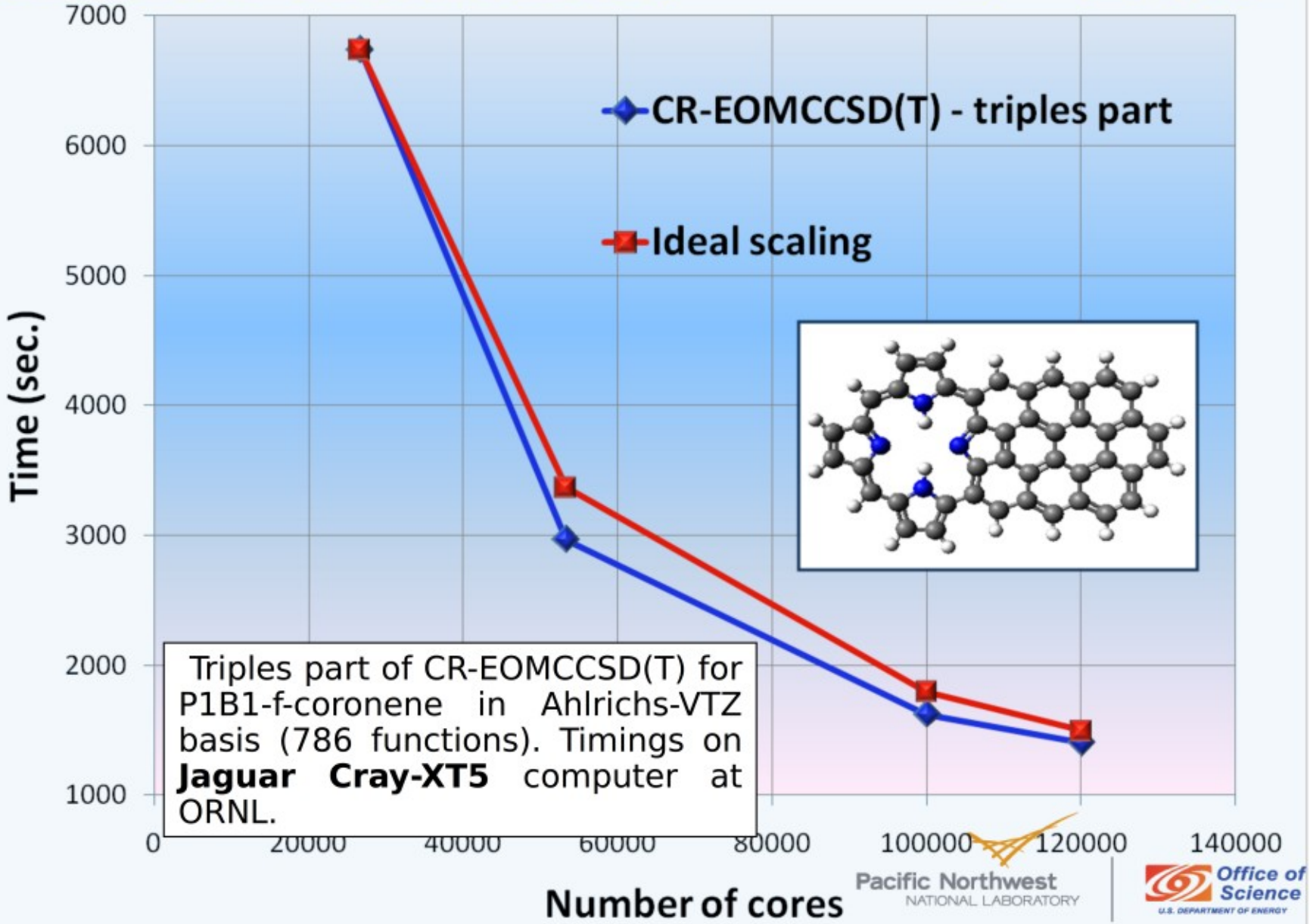

```
next = NXTASK(nprocs, 1)
DO p3b = noab+1,noab+nvab
DO p4b = p3b,noab+nvab
DO h1b = 1,noab
DO h2b = h1b,noab
IF (next.eq.count) THEN
CALL GET_HASH_BLOCK(d_a,dbl_mb(k_a),dim
- 1 + (noab+nvab) * (h1b_1 - 1 + (noab+
+nvab) * (p3b_1 - 1)))))
CALL GET_HASH_BLOCK_I(d_a,dbl_mb(k_a),d
```

Pacific Northwest NATIONAL LABORATORY

Office of Science U.S. DEPARTMENT OF ENERGY

# Parallel performance (Karwolski et al., PNNL)

EMSL




Pacific Northwest NATIONAL LABORATORY

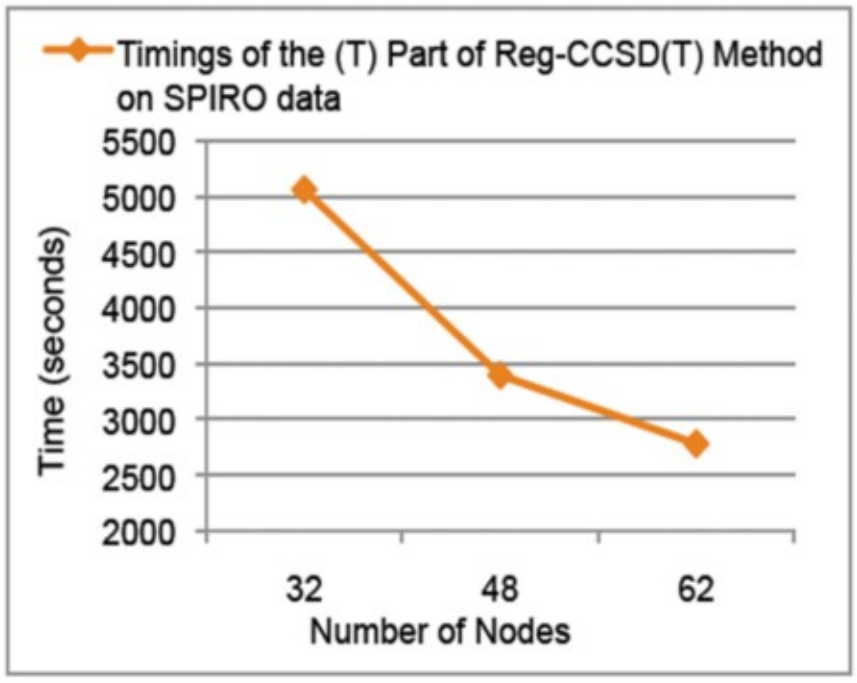
Office of Science U.S. DEPARTMENT OF ENERGY

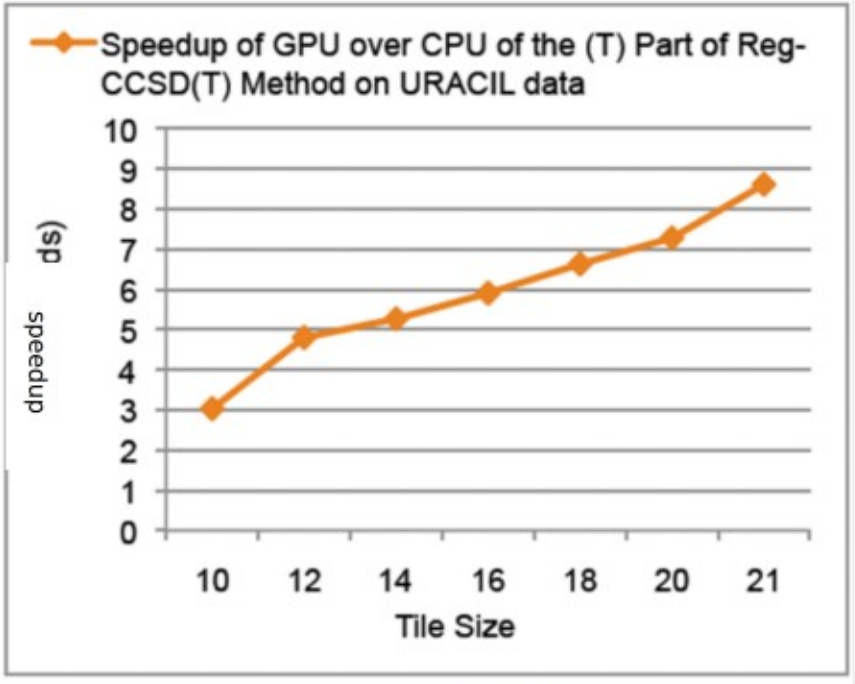
# Towards future computer architectures (Villa,Krishnamoorthy, Kowalski)

EMSL

The CCSD(T)/Reg-CCSD(T) codes have been rewritten in order to take advantage of GPGPU accelerators

Preliminary tests show very good scalability of the most expensive N7 part of the CCSD(T) approach




Pacific Northwest NATIONAL LABORATORY

Office of Science U.S. DEPARTMENT OF ENERGY