NWChem and MADNESS
A brief exploration for exascale

Robert J. Harrison

Institute for Advanced Computational Science
Stony Brook University

and

Center for Scientific Computing
Brookhaven National Laboratory

robert.harrison@stonybrook.edu

BROOKHAVEN
NATIONAL LABORATORY

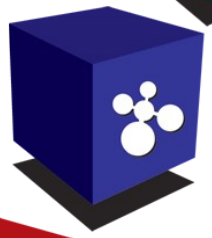
National Science Foundation
WHERE DISCOVERIES BEGIN

Stony Brook University

Molecular Science Software Project

MS³
MOLECULAR SCIENCE SOFTWARE SUITE

ECCE
EXTENSIBLE COMPUTATIONAL CHEMISTRY ENVIRONMENT



NWCHEM
HIGH-PERFORMANCE COMPUTATIONAL CHEMISTRY SOFTWARE

GA TOOLS
PARALLEL COMPUTING LIBRARIES AND SOFTWARE TOOLS

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

(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

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

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

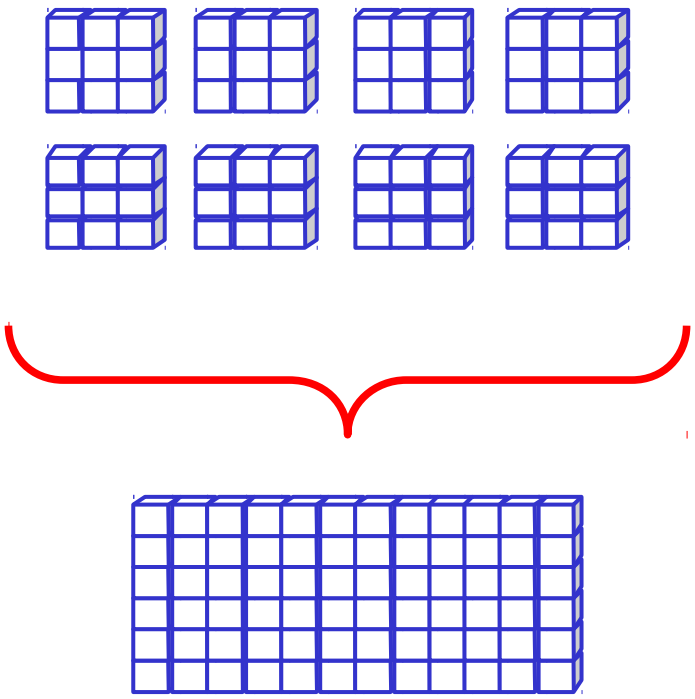
1 integral contributes to 6 Fock Matrix elements

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

- 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/

SC94

Global Arrays: A Portable "Shared-Memory" Programming Model for Distributed Memory Computers

Jaroslaw Nieplocha, Robert J. Harrison and Richard J. Littlefield

Pacific Northwest Laboratory‡, P.O. Box 999, Richland WA 99352

Abstract

Portability, efficiency, and ease of coding are all important considerations in choosing the programming model for a scalable parallel application. The message-passing programming model is widely used because of its portability, yet some applications are too complex to code in it while also trying to maintain a balanced computation load and avoid redundant computations. The shared-memory programming model simplifies coding, but it is not portable and often provides little control over interprocessor data transfer costs.This paper describes a new approach, called Global Arrays (GA), that combines the better features of both other models, leading to both simple coding and efficient execution. The key concept of GA is that it provides a portable interface through which each process in a MIMD parallel program can asynchronously access logical blocks of physically distributed matrices, with no need for explicit cooperation by other processes. We have implemented GA libraries on a variety of computer systems, including the Intel DELTA and Paragon, the IBM SP-1 (all message-passers), the Kendall Square KSR-2 (a nonuniform access shared-memory machine), and networks of Unix workstations.

chemistry. At the same time, we and our colleagues at the Pacific Northwest Laboratory (PNL) have a short-term goal of developing, within the next three years, a suite of parallel chemistry application codes to be used in production mode for chemistry research at PNL's Environmental and Molecular Science Laboratory (EMSL) and elsewhere. The programming model and implementations described here have turned out to be useful for both purposes.

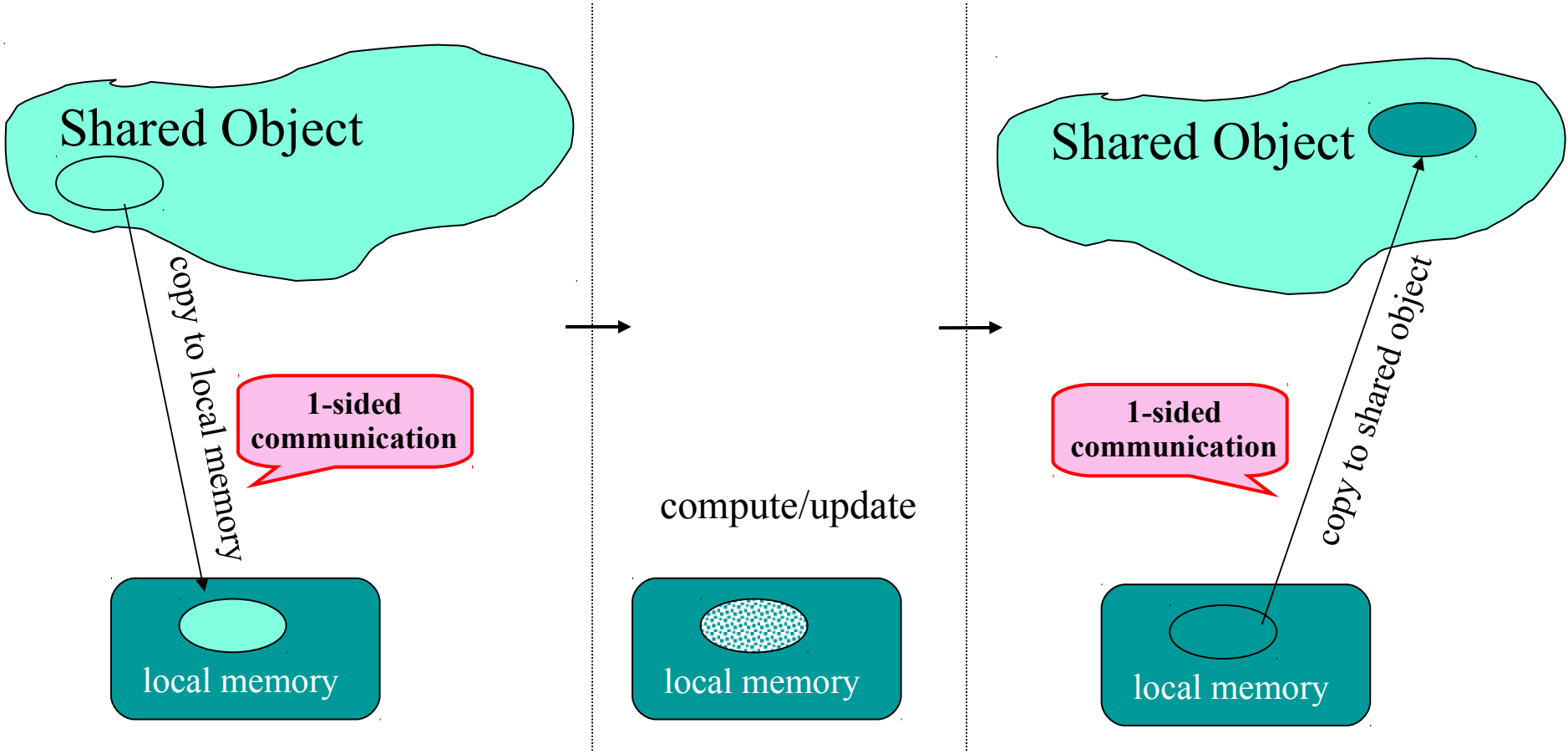
Two assumptions permeate our work. The first is that most high performance parallel computers currently and will continue to have physically distributed memories with Non-Uniform Memory Access (NUMA) timing characteristics, and will thus work best with application programs that have a high degree of locality in their memory reference patterns. The second assumption is that extra programming effort is and will continue to be required to construct such applications. Thus, a recurring theme in our work is to develop techniques and tools that allow applications with explicit control of locality to be developed with only a tolerable amount of extra effort.

There are significant tradeoffs between the important

History and Design

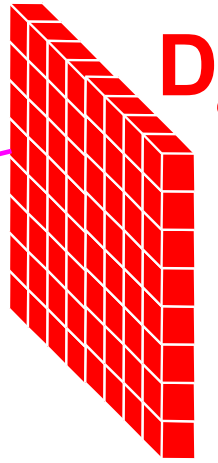
- Prototyping at very start of NWChem project
  - Model full application not just kernel
  - 80-20 rule – more like 90-10 rule
- GA designed to solve a problem
  - Distributing large data structures while supporting irregular computation
  - Entire HF code
  - First 2 attempts (Linda-like) worked for kernel but not the rest of the code

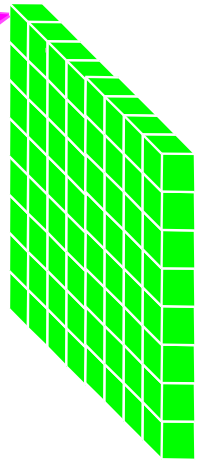
Non-uniform memory access model of computation



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()
```





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

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

Higher-performance code

- Looks nothing like that!
- Sort shell pairs to evaluate in similar batches
  - Precomputation, vectorization – 10-fold speedup
  - Big increase in complexity and memory use
- Integral evaluation code – 100K lines!
- Careful screening with rigorous inequalities
  - Robustness, minimize overhead
-

Highest-performance code

- Looks nothing like that!
- Strives for near linear scaling
- Coulomb interaction
  - Mix of FMM, FFT, and other fast methods
  - (near) linear scaling with system size
- Exchange interaction
  - Heavy screening and physical thresholding
- And this is just 1% of NWChem functionality

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

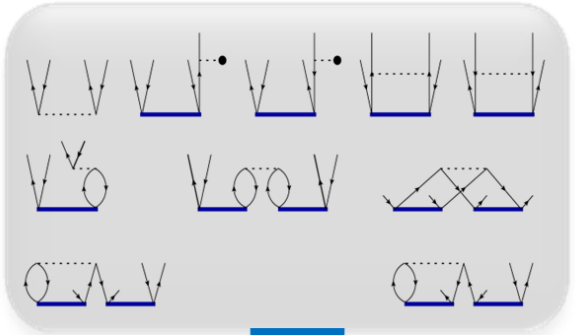
Tensor Contraction Engine (TCE) (Kowalski, PNNL)

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



OCE

$$+\frac{1}{4} v_{ef}^{mn} t_{ij}^{ef} t_{mn}^{ab} - \frac{1}{2} v_{ef}^{mn} t_{mi}^{ef} t_{nj}^{ab} +$$

TCE

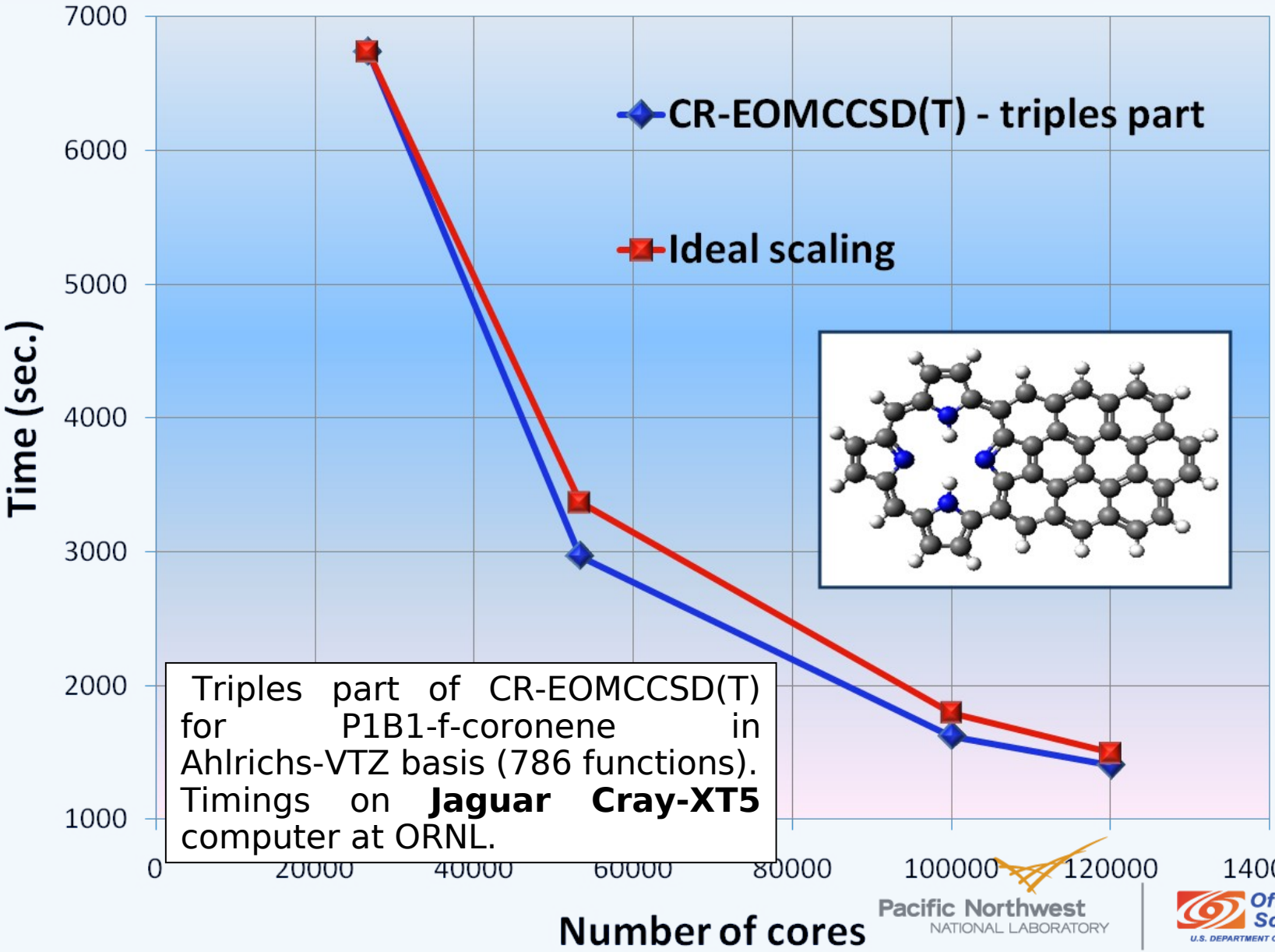
```
next = NXTASK(nprocs, 1)
DO p3b = noab+1,noab+nvab
DO p4b = p3b,noab+nvab
DO h1b = 1,noab
DO h2b = h1b,noab
IF (next.eq.count) THEN
CALL GET_HASH_BLOCK(d_a,dbl_mb(k_a),dim
- 1 + (noab+nvab) * (h1b_1 - 1 + (noab+
+nvab) * (p3b_1 - 1)))))
CALL GET_HASH_BLOCK_I(d_a,dbl_mb(k_a),d
```

Pacific Northwest NATIONAL LABORATORY

Office of Science U.S. DEPARTMENT OF ENERGY

Parallel performance (Karwolski et al., PNNL)

EMSL



Triples part of CR-EOMCCSD(T) for P1B1-f-coronene in Ahlrichs-VTZ basis (786 functions). Timings on **Jaguar Cray-XT5** computer at ORNL.

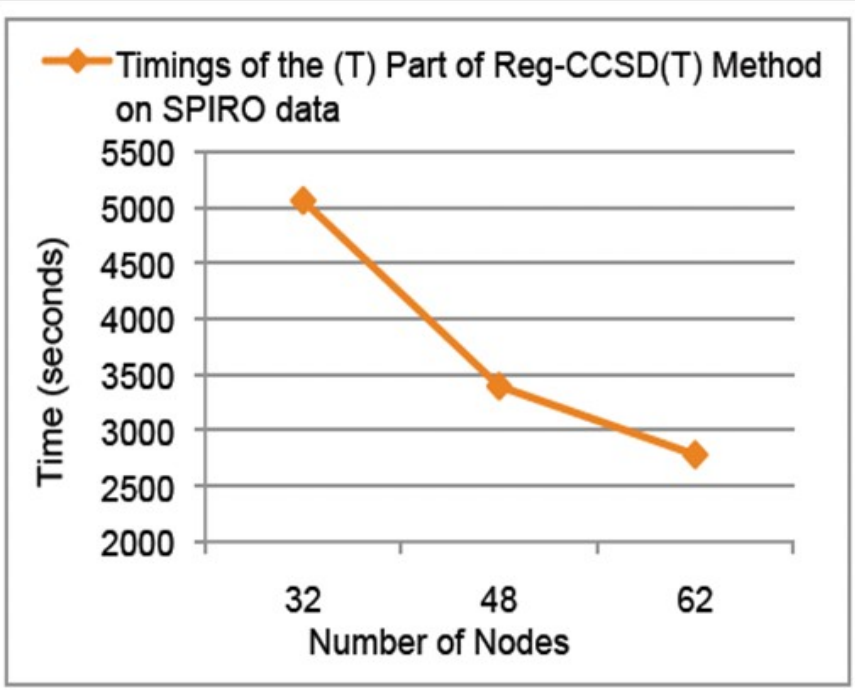
Pacific Northwest NATIONAL LABORATORY

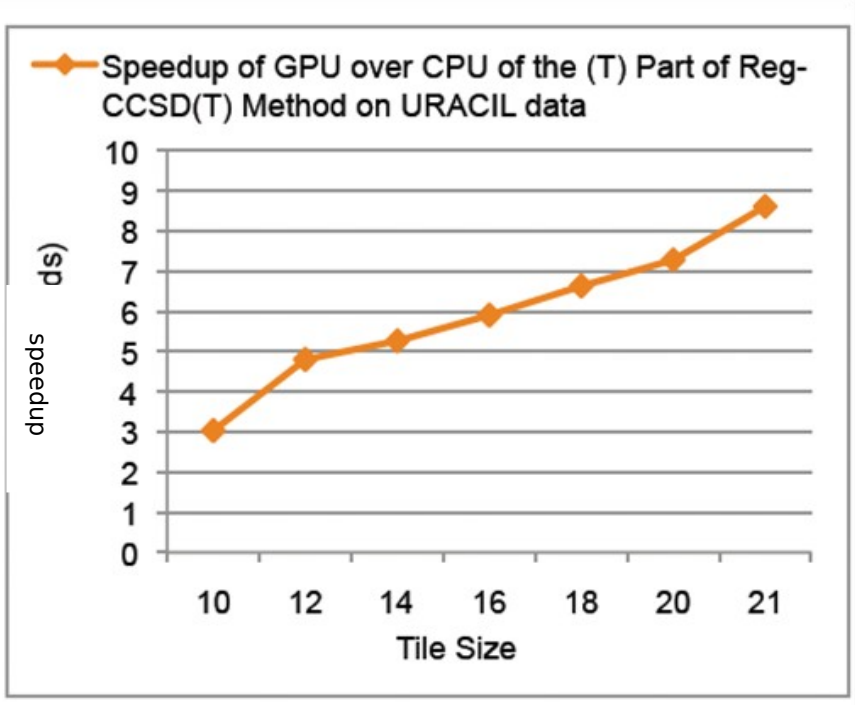
Office of Science U.S. DEPARTMENT OF ENERGY

Towards future computer architectures (Villa,Krishnamoorthy, Kowalski)

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

MADNESS

Multiresolution
Adaptive
Numerical
Scientific
Simulation

Multiresolution Adaptive Numerical Scientific Simulation

George I. Fann[1], Diego Galindo[1], Robert J. Harrison[3], Scott Thornton[2], Judy Hill[1], and Jun Jia[1]

[1]Oak Ridge National Laboratory
[2]University of Tennessee, Knoxville
[3]Stony Brook University, Brookhaven National Laboratory

In collaboration with

Gregory Beylkin[4], Lucas Monzon[4], Hideo Sekino[5] and Edward Valeev[6]

[4]University of Colorado
[5]Toyohashi Technical University, Japan
[6]Virginia Tech

BROOKHAVEN
NATIONAL LABORATORY

robert.harrison@gmail.com

National Science Foundation
WHERE DISCOVERIES BEGIN

Stony Brook University

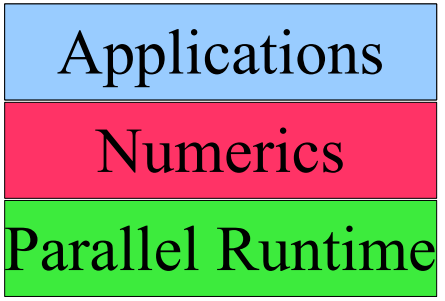
Big picture

- 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 ε
- Semantic gap
  - Why are our equations just O(100) lines but programs O(1M) lines?
- Facile path from laptop to exaflop

What is MADNESS?

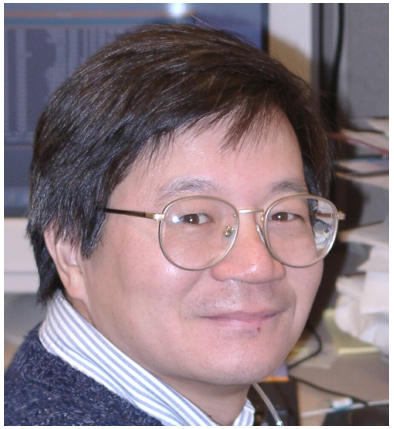
- A general purpose numerical environment for reliable and fast scientific simulation
  - Chemistry, nuclear physics, atomic physics, material science, nanoscience, climate, fusion, ...
- A general purpose parallel programming environment designed for the peta/exa-scales
- Addresses many of the sources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



IBM BGQ Team

- ANL
  - Alvaro Vasquez
  - Jeff Hammond
  - Nichols Romero
- OSU
  - Kevin Stocks
- SBU
  - Robert Harrison
- UTK
  - Scott Thornton



George Fann

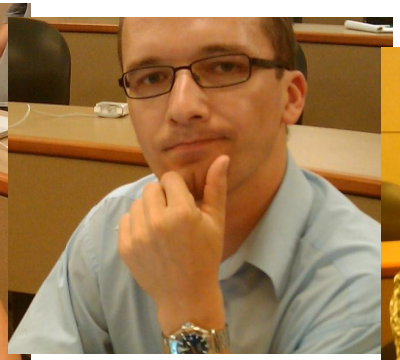
Judy Hill

Gregory Beylkin

Rebecca Hartman-Baker

Jun Jia
Tetsuya Kato
Justus Calvin
J. Pei

Ariana Beste



Eduard Valeyev

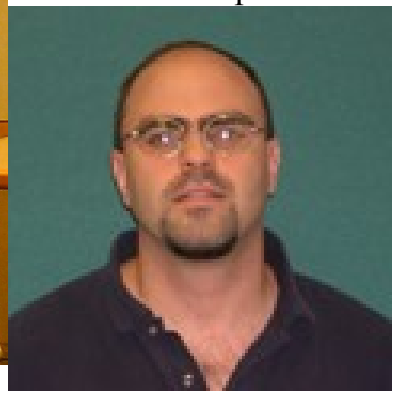
Alvaro Vasquez

Hideo Sekino

Robert Harrison

Nicholas Vence

Takahiro Ii



Scott Thornton

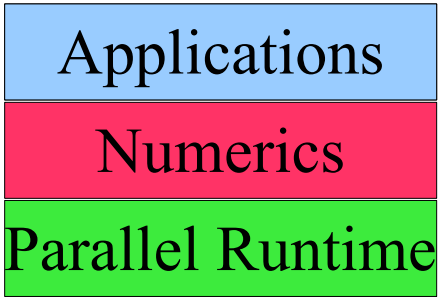
Matt Reuter

Paul Sutter

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

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

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

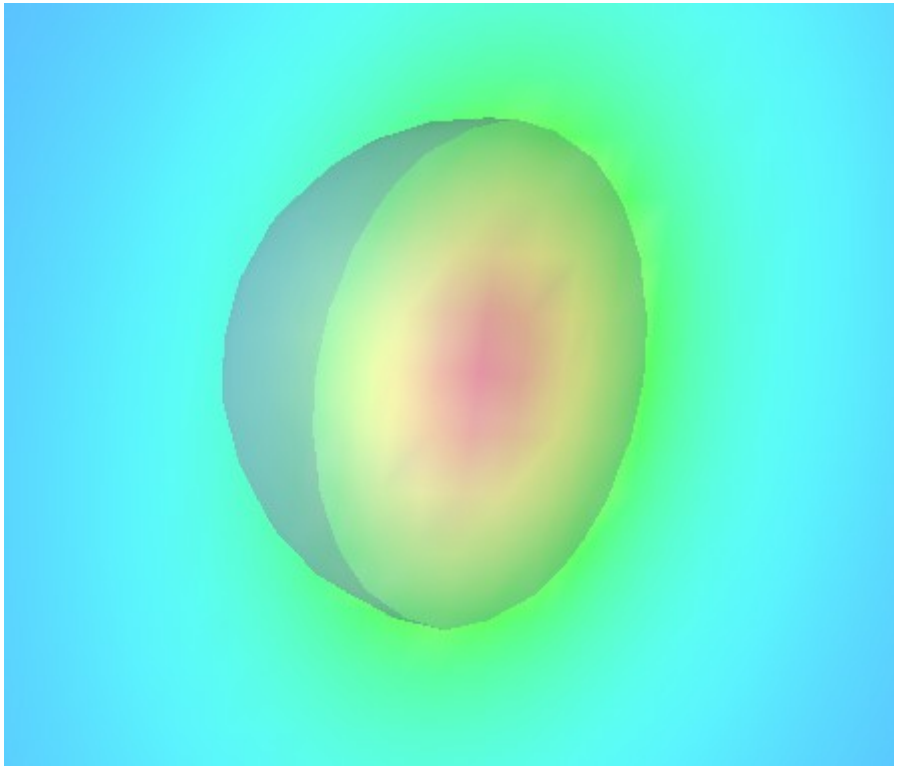
$$u = \nabla^{-2}\left(-4 * \pi * f\right)$$

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", λ

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

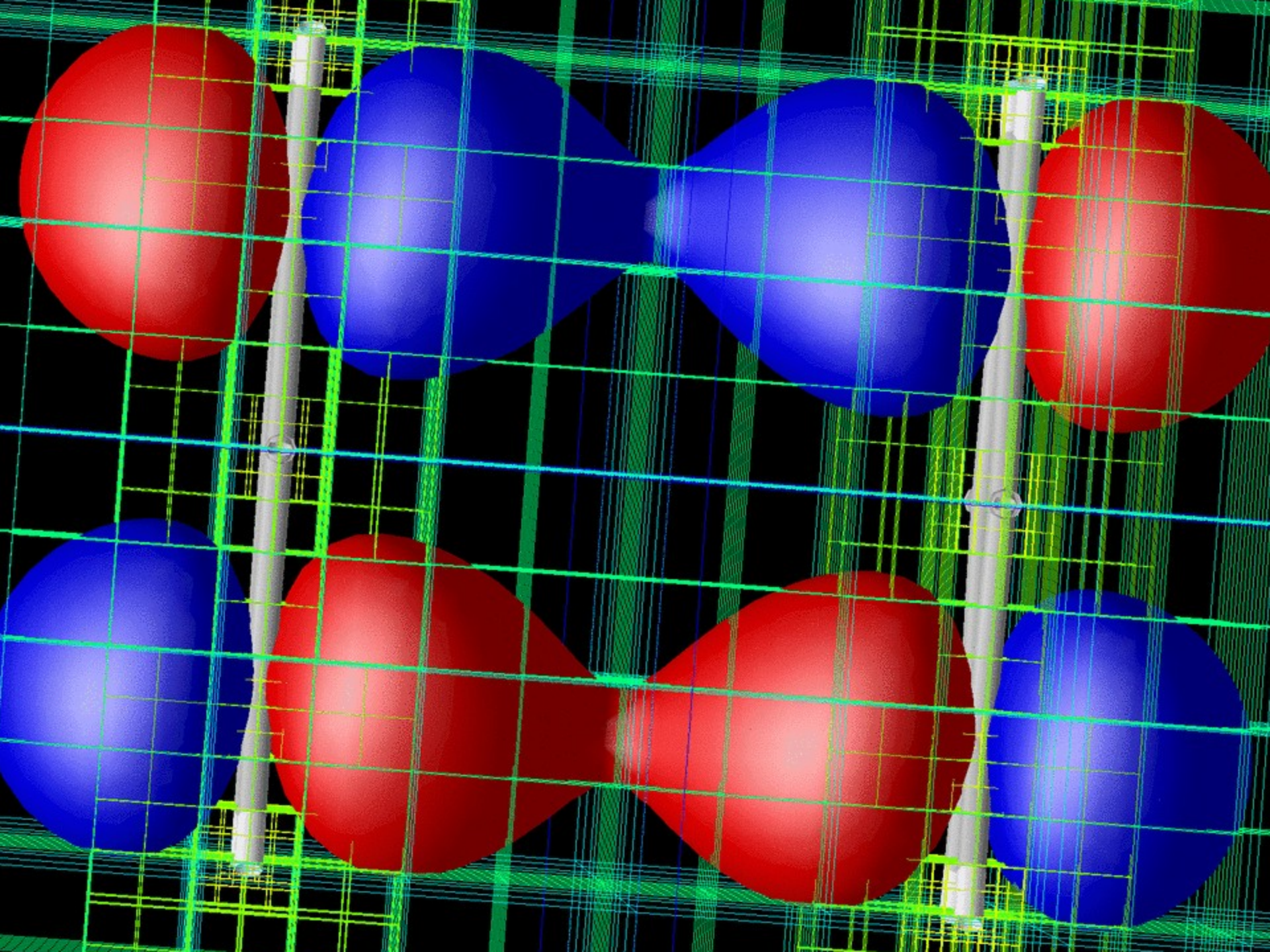
$$f(x_1, \ldots, x_n) = \sum_{l=1}^{M} \sigma_l \prod_{i=1}^{d} f_i^{(l)}(x_i) + O(\epsilon)$$

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

- Low-operator rank

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

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



Three equivalent representations

- Scaling function basis (reconstructed)

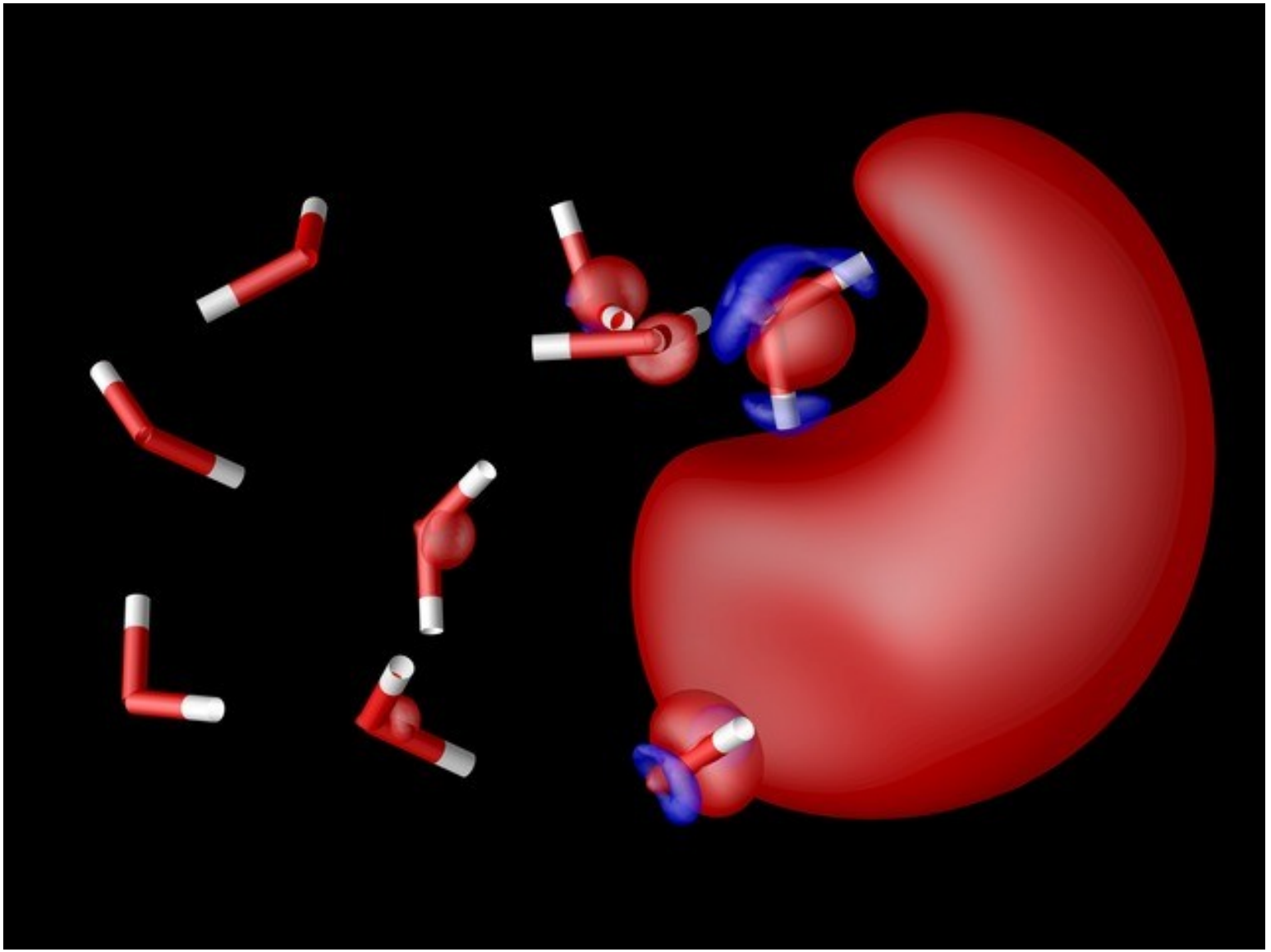
$$f^n(x) = \sum_{l=0}^{2^n-1} \sum_{i=0}^{k-1} s_{il}^n \phi_{il}^n(x)$$

- Multi-wavelet basis (compressed)

$$f^n(x) = \sum_{i=0}^{k-1} s_{i0}^0 \phi_{i0}^0(x) + \sum_{n'=0}^{n-1} \sum_{l=0}^{2^{n'}-1} \sum_{i=0}^{k-1} d_{il}^{n'} \psi_{il}^{n'}(x)$$

- Rapid compression/reconstruction
- Values at Gauss-Legendre points in each box
- Use appropriate basis for a given operation

Molecular Electronic Structure



Energy and gradients

ECPs coming (Sekino, Kato)

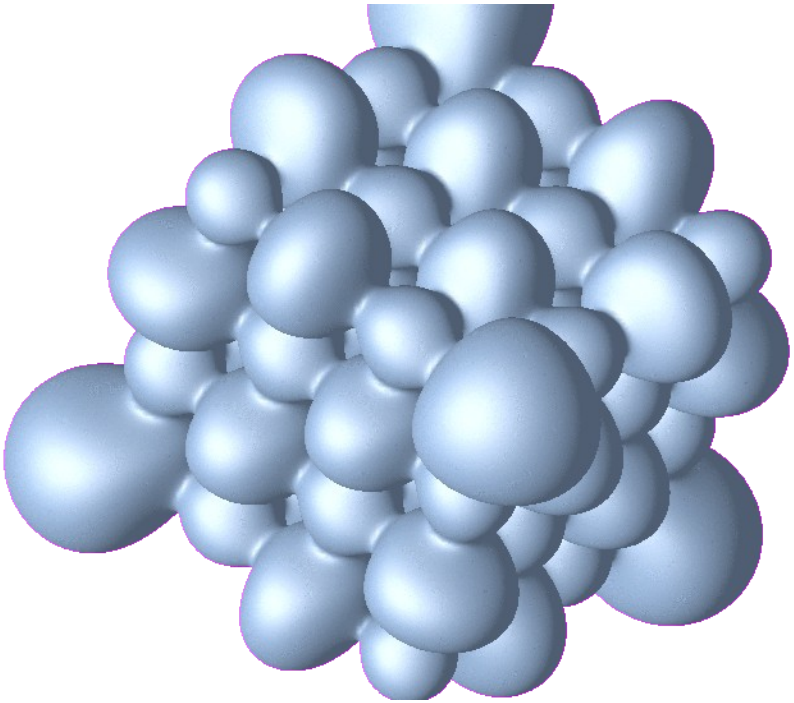
Response properties (Vasquez and Sekino)

Still not as functional as previous Python version

Spin density of solvated electron

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

Nuclear physics

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

UNEDF Deformed SLDA in 3-D
deformation in an external trap with aspect ratio of 1/16

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

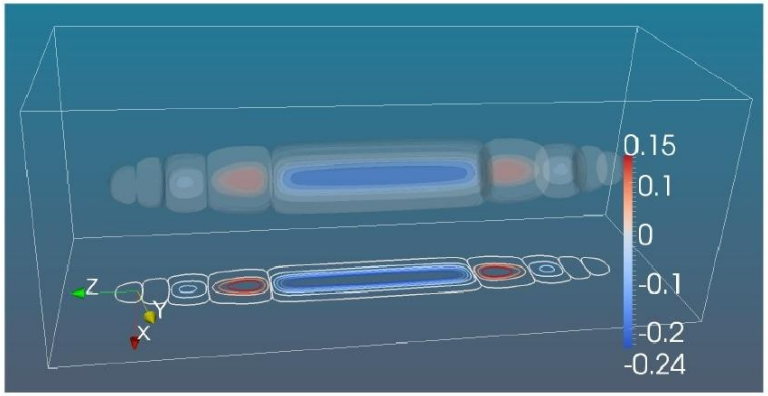
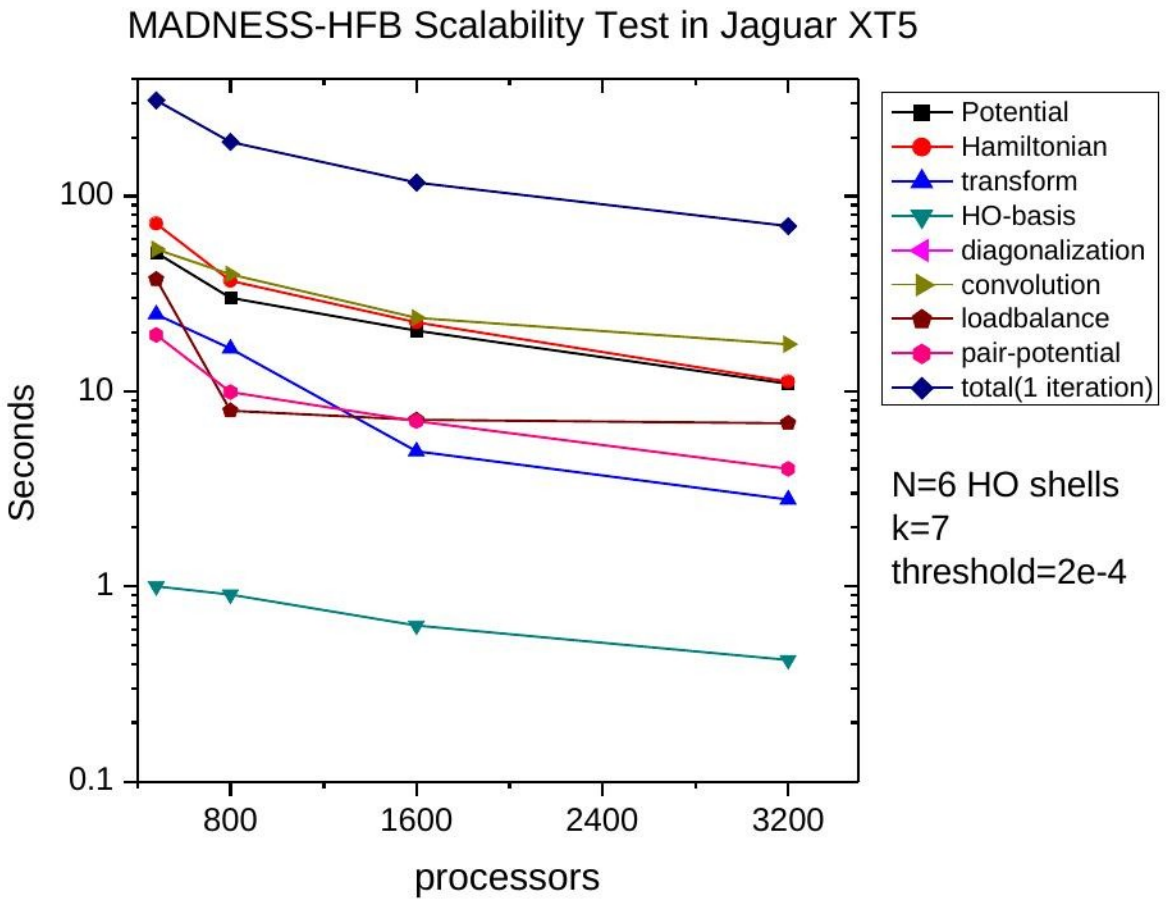


Figure 4. Pairing density $\kappa(x, y, z)$, calculated by MADNESS-HFB for the elongated trap wit $\eta = 16$. The box scale in view is x[-12, 12], y[-12, 12] and z[-32, 32].

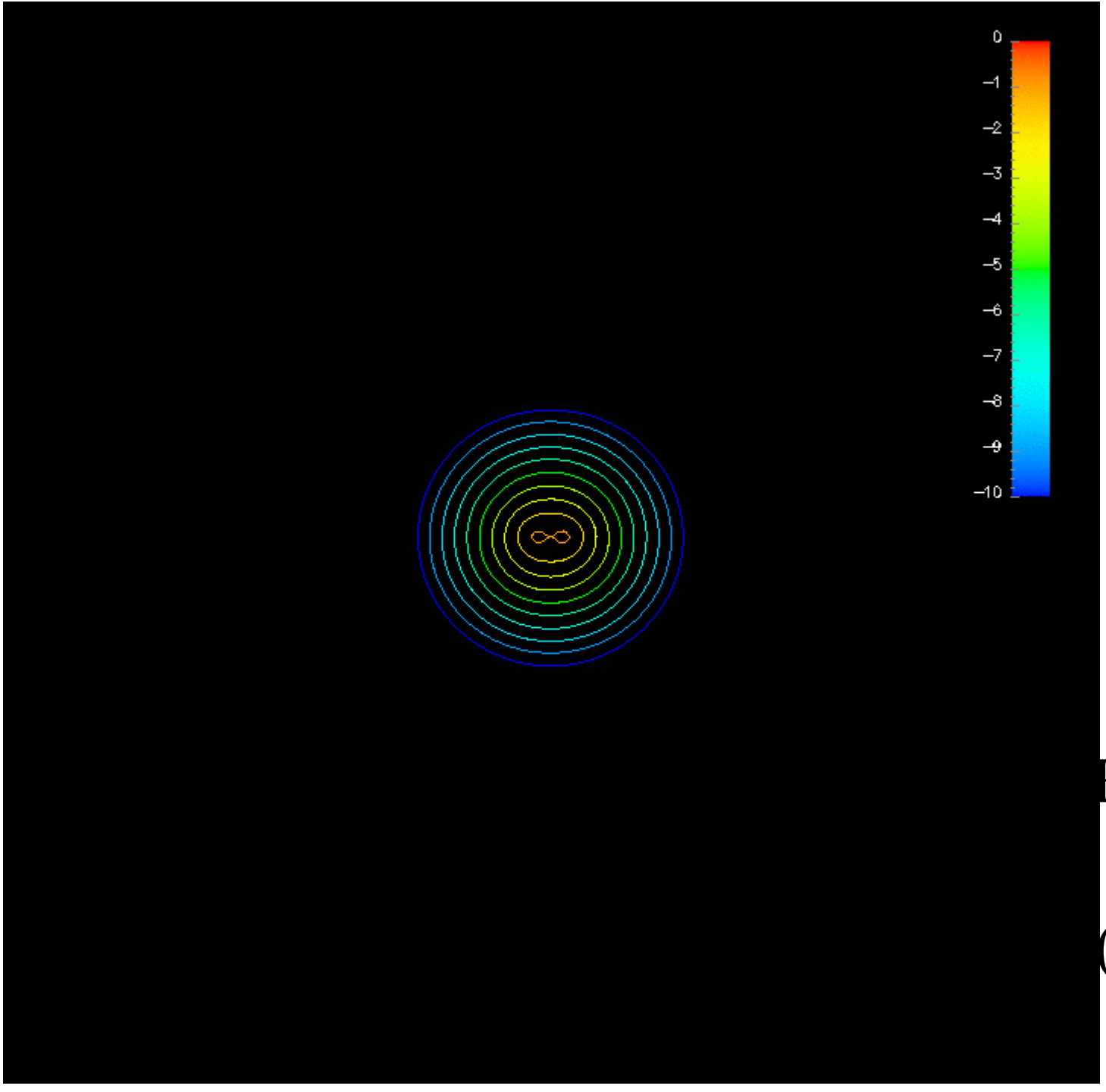
BGQ Early Science Project Activities

- We are testing a new linear-response module to solve TDDFT equations.
- Molecular properties (dipole polarizabilities, NMR chemical shifting, etc.)
- Support for well known hard pseudopotentials (i.e. Krack, Goedecker,etc.)
- Speedup of Hartree-Fock exchange evaluations via screening parameters.
- Implementation of new DFT functionals.
- Improving parallel scalability for current supercomputer architectures.

UNEDF Solving HFB in Production



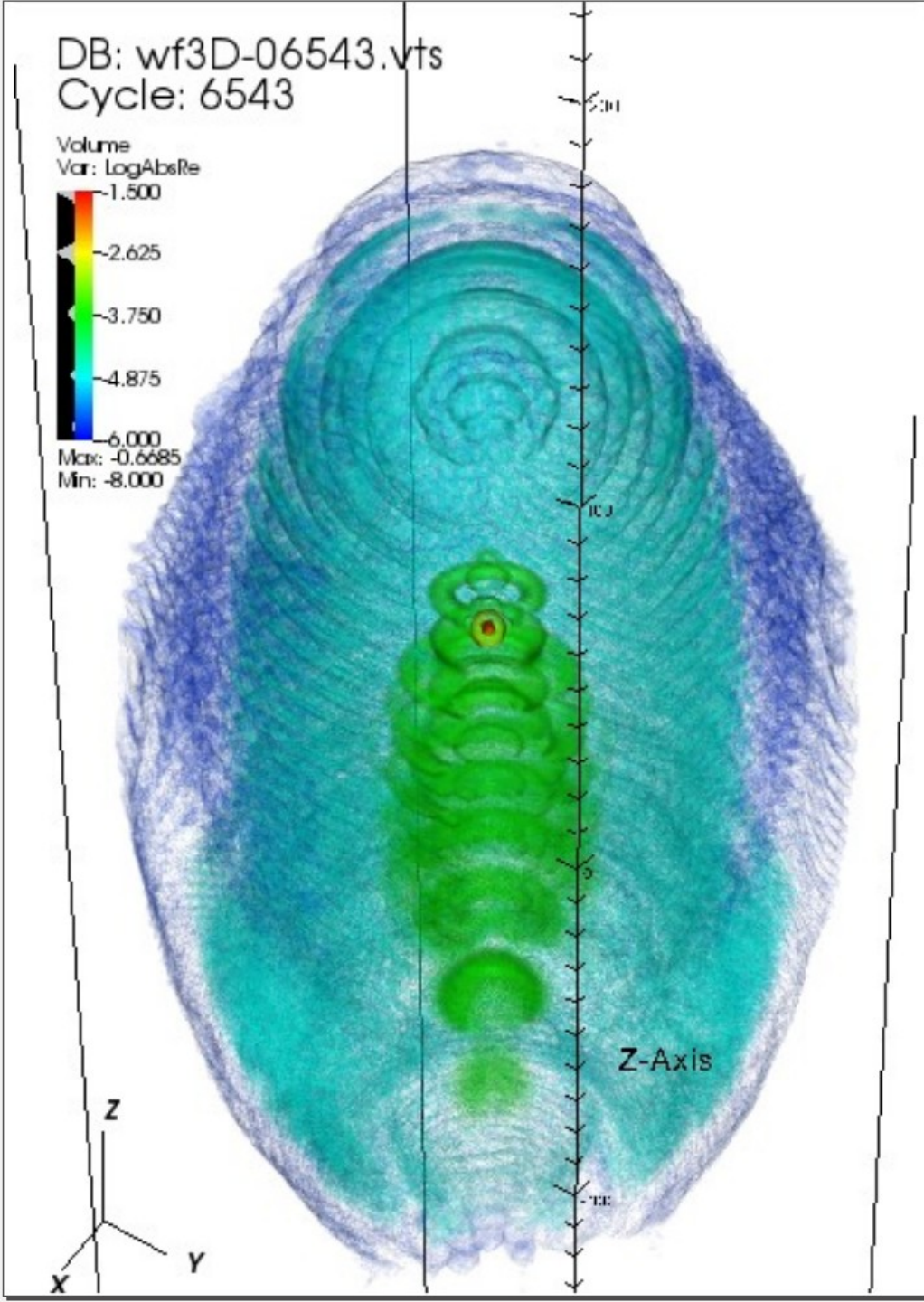
Realizable scaling even for small problems in 3-D



Time dependent electronic structure

Vence, Krstic, Harrison UT/ORNL

H_2^+ molecule in laser field (fixed nuclei)

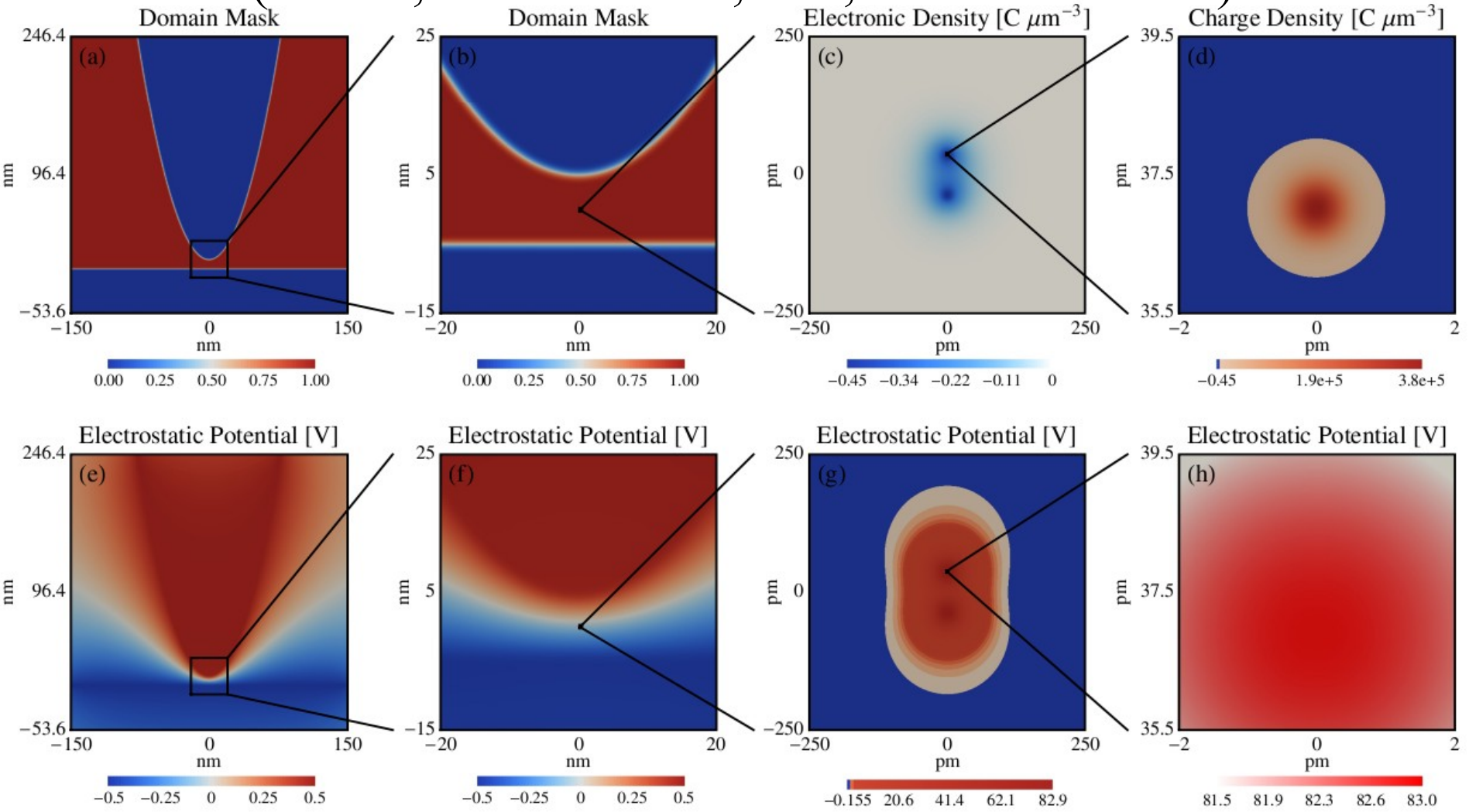


H-atom IR pulse

The eletric field repeatedly rips the electron out ~300 Bohr and brings it back to rescatter off the nucleus.

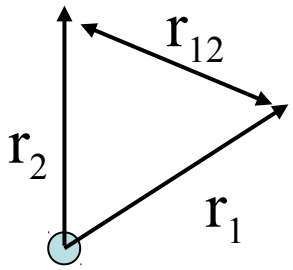
Nanoscale photonics

(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

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

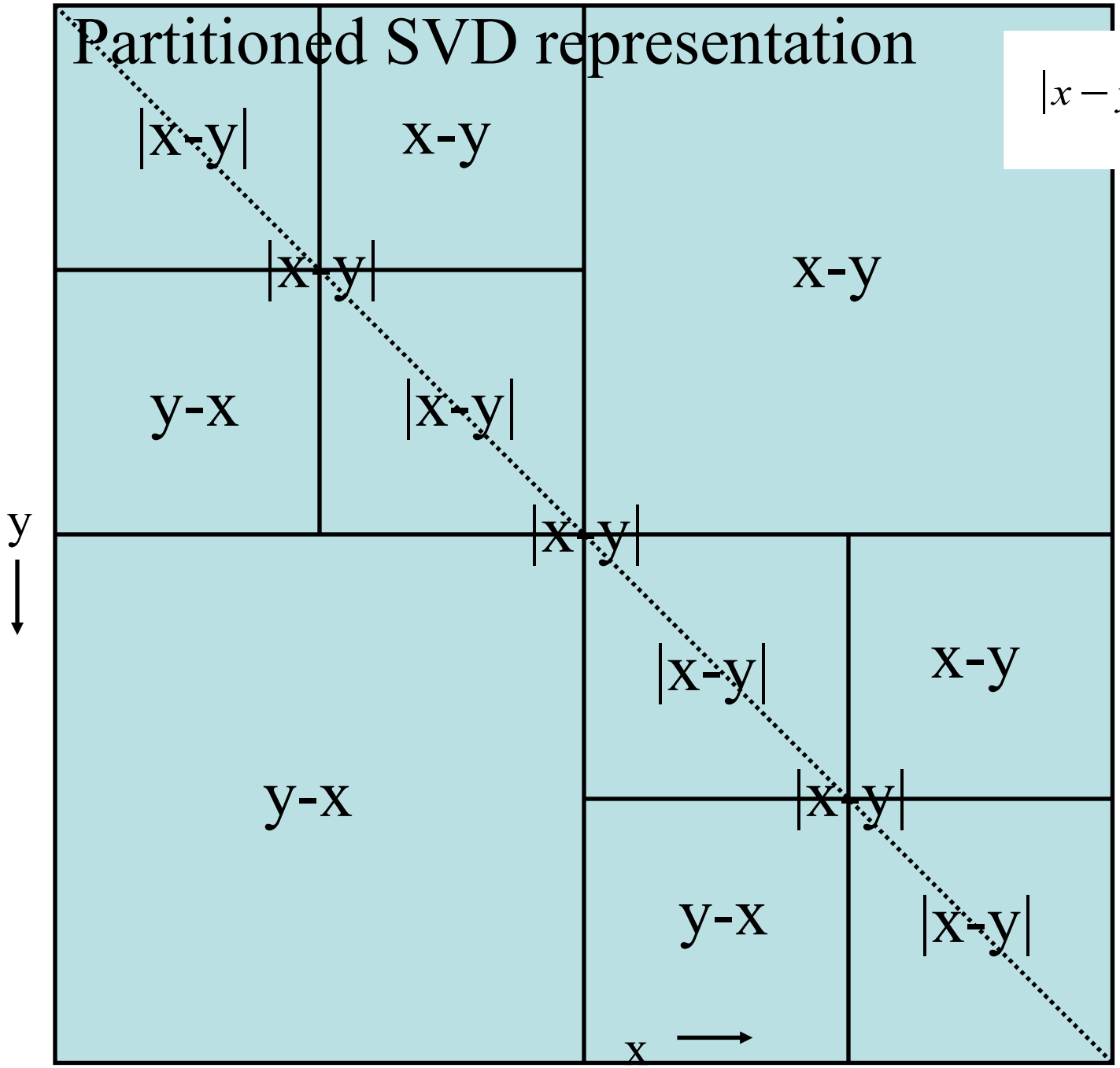
- Include the inter-electron distance in the wavefunction
  - E.g., Hylleraas 1938 wavefunction for He

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

  - 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

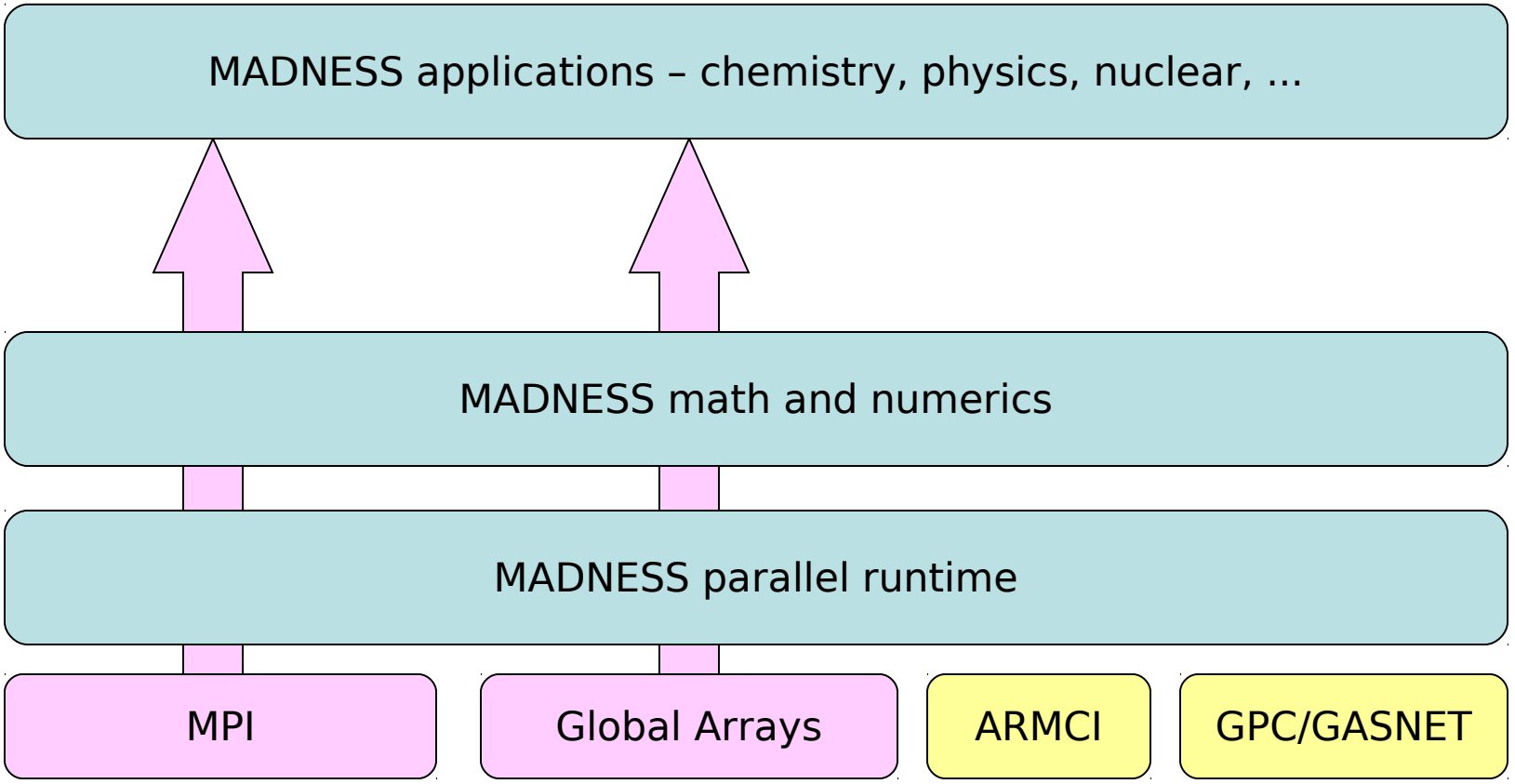
In 3D, ideally must be one box removed from the diagonal

Diagonal box has full rank

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

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

MADNESS architecture



Intel Thread Building Blocks more scalable; also ported to BGQ

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

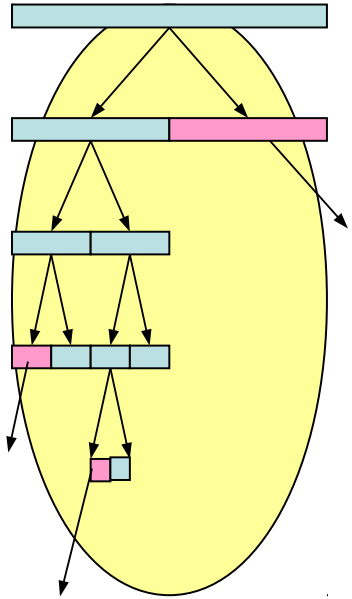
Futures

- 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;
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.

Global Namespaces

- Specialize global names to containers
  - Hash table done
  - Arrays, etc., planned
- 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);
```

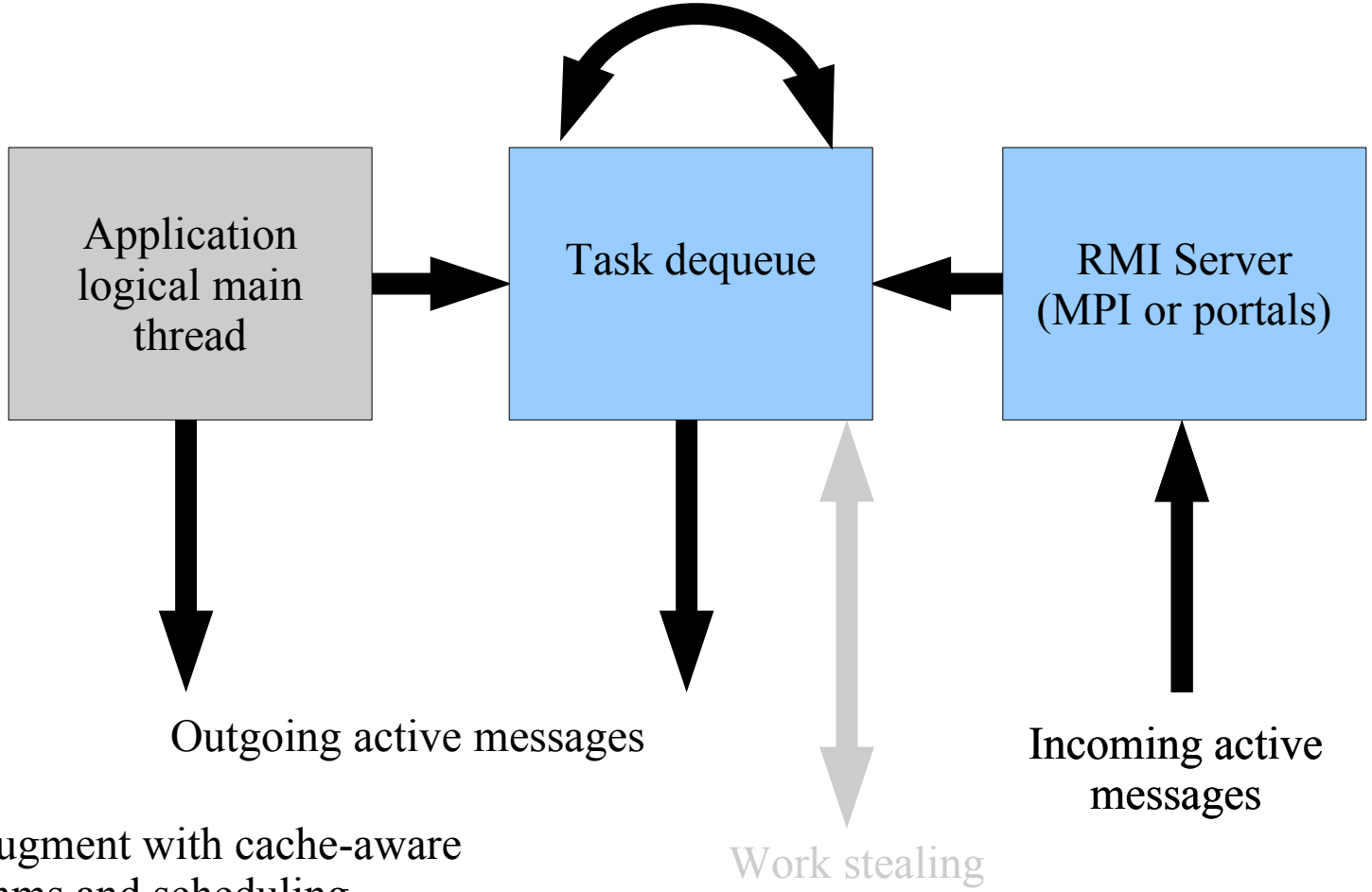
A container is created mapping indices to values.

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

Computational kernels

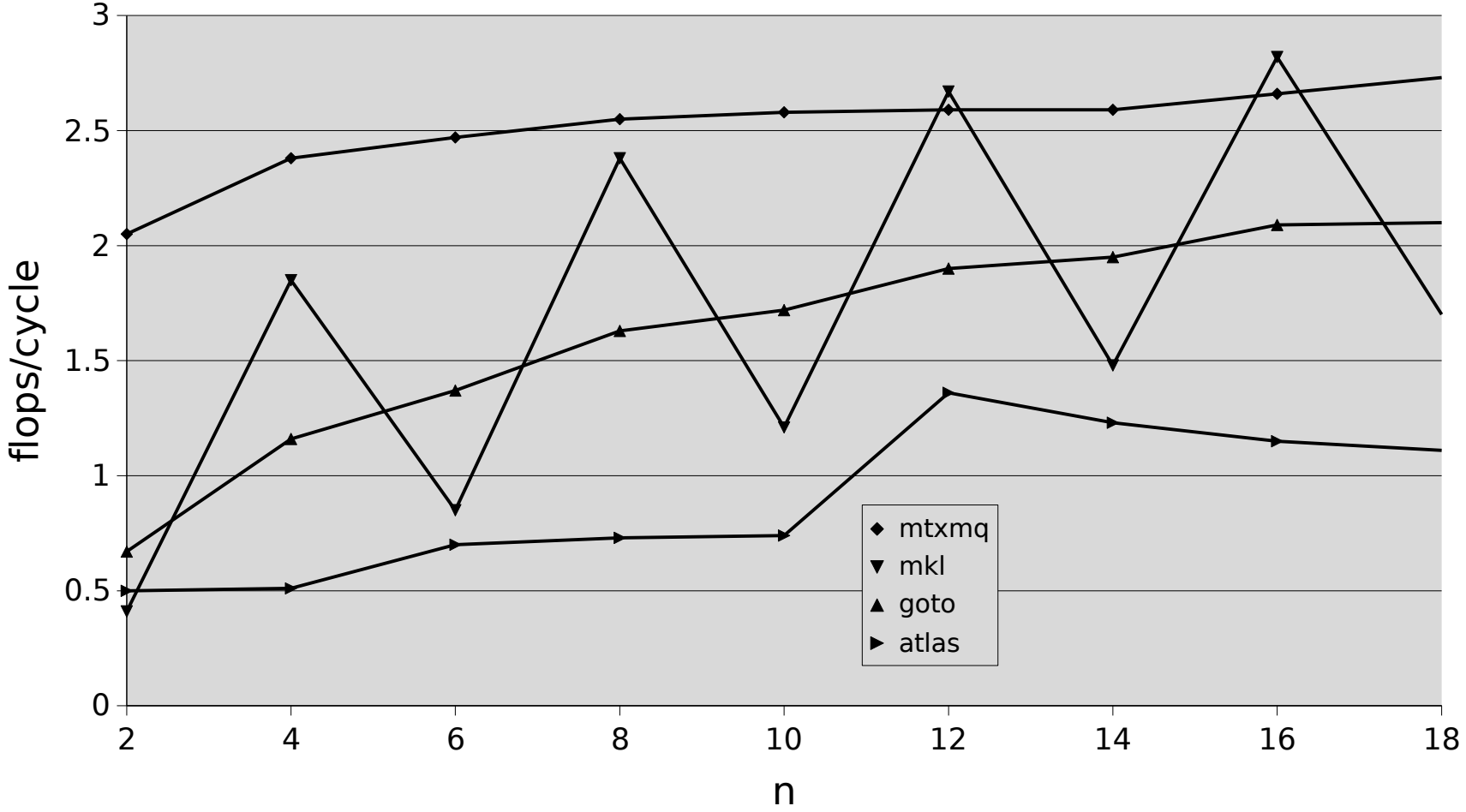
- Discontinuous spectral element
  - In each “box” a tensor product of coefficients
  - Most operations are small matrix-multiplication

$$r_{i'j'k'} = \sum_{ijk} s_{ijk} c_{ii'} c_{jj'} c_{kk'} = \sum_{k} \left(\sum_{j} \left(\sum_{i} s_{ijk} c_{ii'} \right) c_{jj'} \right) c_{kk'}$$

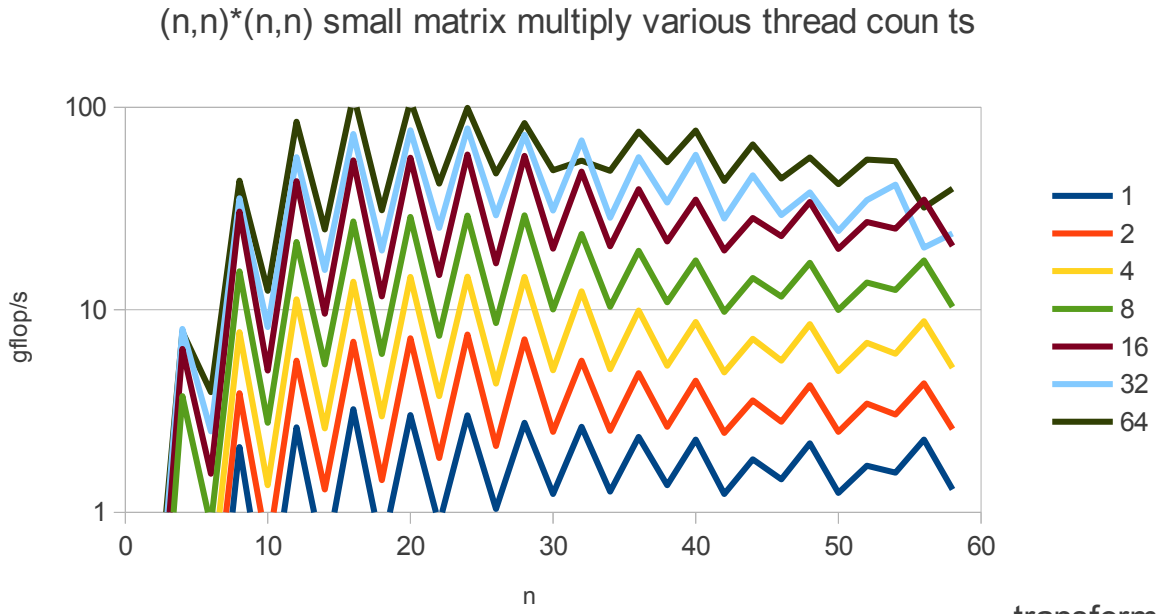
$$\Rightarrow r = \left(\left(s^T c \right)^T c \right)^T c$$

  - Typical matrix dimensions are 2 to 30
  - E.g., $(20,400)^T$ * (20,20)

Comparison with MKL, Goto, ATLAS on Intel Xeon 5355 for $(20,400)^T*(20,n)$.



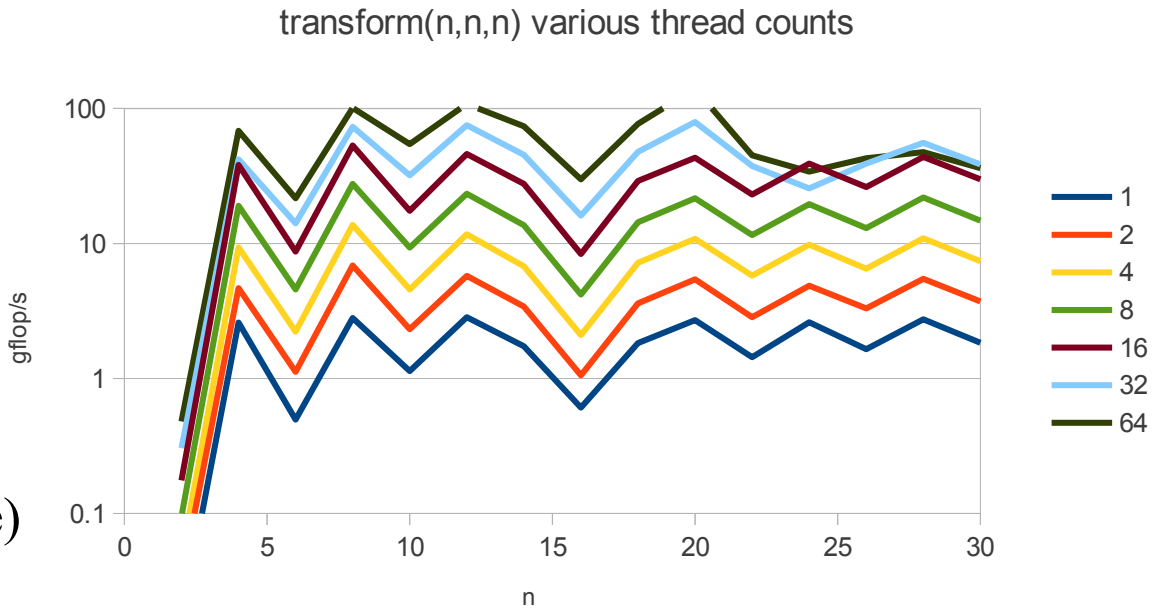
MtXM performance on BGQ



Kevin Stocks OSU

64 threads, best performance is 139.7 GFLOPS (trans(400,20,20))
* Theoretical peak is 204.8
* Linpack is approx. 166.3

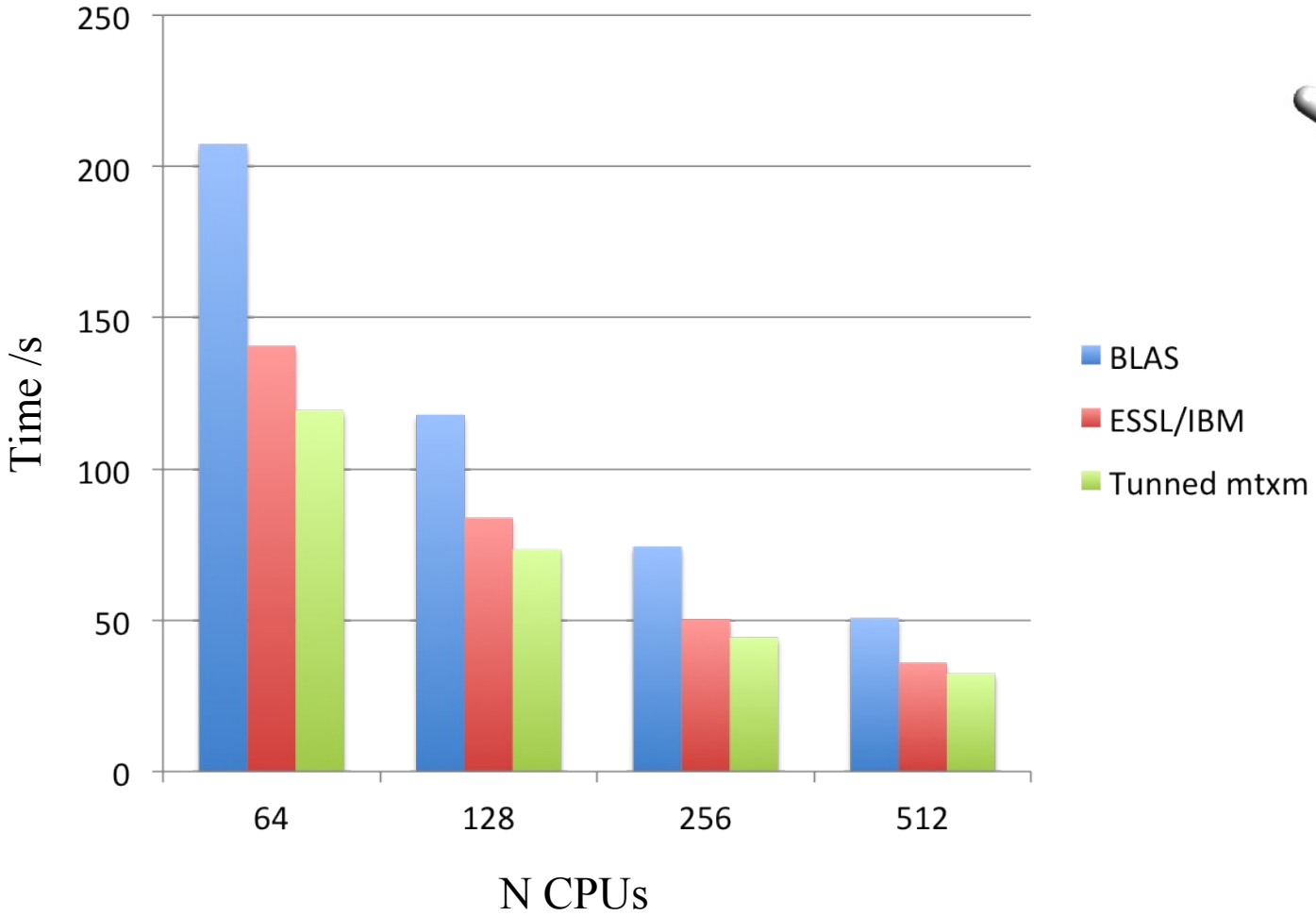
(scaling top500 results to one node)

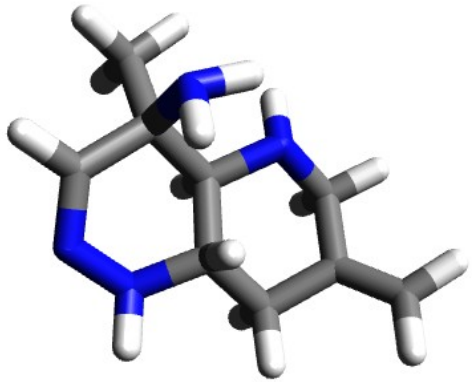


Benefit of tuned mTxm in BG/P Performance

Strong scaling

Molecular system with 13 heavy atoms, DFT, k=8, one iteration





Summary

- Exascale programming models
  - Resilience, Power, Performance, Productivity
  - Productivity is arguably the most important
  - Enable innovation and discovery at scale
- MADNESS and NWChem
  - Frameworks – places for disciplines to meet to leverage investments and expertise
  - Face difference challenges in moving forward

BROOKHAVEN NATIONAL LABORATORY

Stony Brook University