Some alternatives to MPI+OpenMP
A brief foray into MADNESS

Robert J. Harrison
Institute for Advanced Computational Science
robert.harrison@stonybrook.edu

Stony Brook University

iACS
INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

Outline

- There’s more to life than MPI+OpenMP
- What is MADNESS?
  - Why does MADNESS need a new runtime
  - Parallel programming inside MADNESS

Cilk – an algorithmic multithreaded language

- Charles E. Leiserson et al., MIT
  - http://supertech.csail.mit.edu/cilk/
  - http://software.intel.com/en-us/articles/intel-cilk-plus/

Now Cilk++ from Intel

- Programmer responsible only for expressing concurrency
  - run time does scheduling using work stealing
- Adds to C the following keywords

  `spawn, sync, cilk, inlet, abort, shared, private, SYNCHED`
- Primarily shared-memory only

```
#include <stdlib.h>
#include <stdio.h>

int fib (int n)
{
    if (n<2) return (n);
    else
    {
        int x, y;

        x = fib (n-1);
        y = fib (n-2);

        return (x+y);
    }
}

int main (int argc, char *argv[])
{
    int n, result;

    n = atoi(argv[1]);
    result = fib (n);

    printf ("Result: %d\n", result);
    return 0;
}
```

(a)

```
#include <stdlib.h>
#include <stdio.h>

cilk int fib (int n)
{
    if (n<2) return n;
    else
    {
        int x, y;

        x = spawn fib (n-1);
        y = spawn fib (n-2);

        sync;

        return (x+y);
    }
}

cilk int main (int argc, char *argv[])
{
    int n, result;

    n = atoi(argv[1]);
    result = spawn fib(n);

    sync;

    printf ("Result: %d\n", result);
    return 0;
}
```

(b)

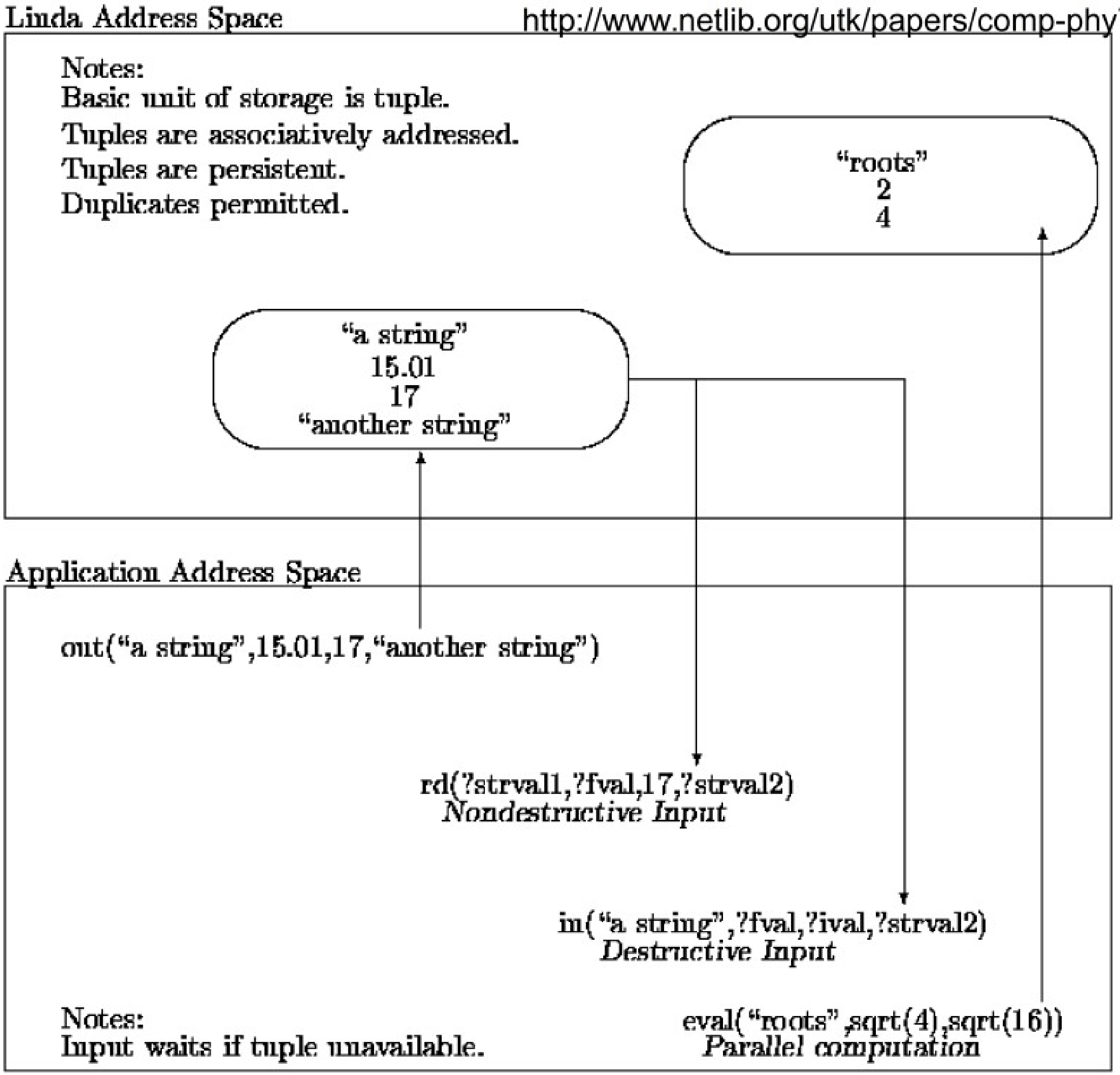
Figure 2.1: (a) A serial C program to compute the nth Fibonacci number. **(b)** A parallel Cilk program to compute the nth Fibonacci number.

Linda – a coordination language

- David Gelernter, Yale
  - Scientific Computing Associates, Inc.
  - Nicholas Carriero and David Gelernter, “How to write parallel programs – a first course”, MIT press
  - http://www.cs.uwaterloo.ca/~fmavadda/p444-carriero.pdf
  - http://www.lindaspaces.com
- Tuple space – a logically shared space
  - Tuple – an ordered list of elements
    - (99,”fred”,3.14)
- Adds to C the following keywords

```
in, rd, out, eval
```

http://www.netlib.org/utk/papers/comp-phy7/node3.html


Linda Address Space
Notes:
Basic unit of storage is tuple.
Tuples are associatively addressed.
Tuples are persistent.
Duplicates permitted.
"roots" 2 4
"a string" 15.01 17 "another string"
Application Address Space
out("a string",15.01,17,"another string")
rd(?strval1,?fval,17,?strval2)
Nondestructive Input
in("a string",?fval,?ival,?strval2)
Destructive Input
Notes:
Input waits if tuple unavailable.
eval("roots",sqrt(4),sqrt(16))
Parallel computation

Map Reduce

- More efficient use of bandwidth and traversal of remote (disk resident) data structures
- Primitive functional programming
- Inefficient without intelligent runtime and optimizing expression engine
- r=map(f,v)
  - for (i=0; i<n; i++) r[i] = f(v[i])
- sum=reduce(op,v)
  - for (i=0; i<n; i++) op(sum,v[i])

Active Messages

- David E. Culler, Berkeley (Cornell)
  - von Eicken, "Active Messages: an Efficient Communication Architecture for Multiprocessors," http://www.cs.cornell.edu/home/tve/thesis/
- Lightweight remote procedure call
  - Envisioned as enabling compilers with
  - In part motivated by CM-5 and NCube
  - Handlers

Continuation Passing

- This is how MADNESS works
  - C.f., Cilk, HPCS languages, Charm++
- http://en.wikipedia.org/wiki
  - Continuation-passing_style
  - Continuation
- Some of the benefits of a functional style
  - But ease of programming require side effects

Charm++

An object-oriented, asynchronous, message passing, parallel, programming paradigm.

- Laximant Kale, UIUC
  - http://charm.cs.illinois.edu
- A Charm++ program is composed in terms of objects that communicate via messages
  - In OO speak message == method invocation
  - Chare – an object with state that can send/receive messages to/from other chares
  - Chare array – a globally addressable name space for chares
  - Parallel run time responsible for placement and scheduling – underlying H/W invisible

PGAS

- Partitioned global address space
- Co-array fortran
  - Now in Fortran standard (for better or worse)
  - http://www.co-array.org/
- UPC
  - Now in GCC and on IBM & Cray
  - http://upc.gwu.edu/

HPCS Languages

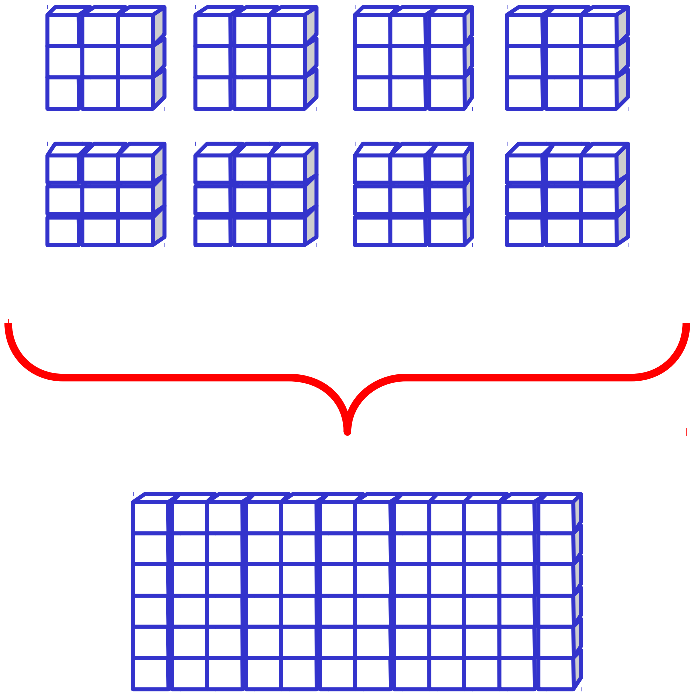
- Chapel
  - http://chapel.cray.com/
- Fortress
  - http://projectfortress.sun.com/Projects/Community/
- X10
  - http://x10-lang.org/
- Evaluation with quantum chemistry app
- http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.146.2715

Performance of parallel algorithms

- Speedup
  - T(0)/T(P)
- Efficiency
  - T(0)/T(P)/P
- Isoefficiency function
  - Problem size as function of P and efficiency
  - Weak vs. strong scaling
- Amdahl’s law

Global Arrays (technologies)

Physically distributed data



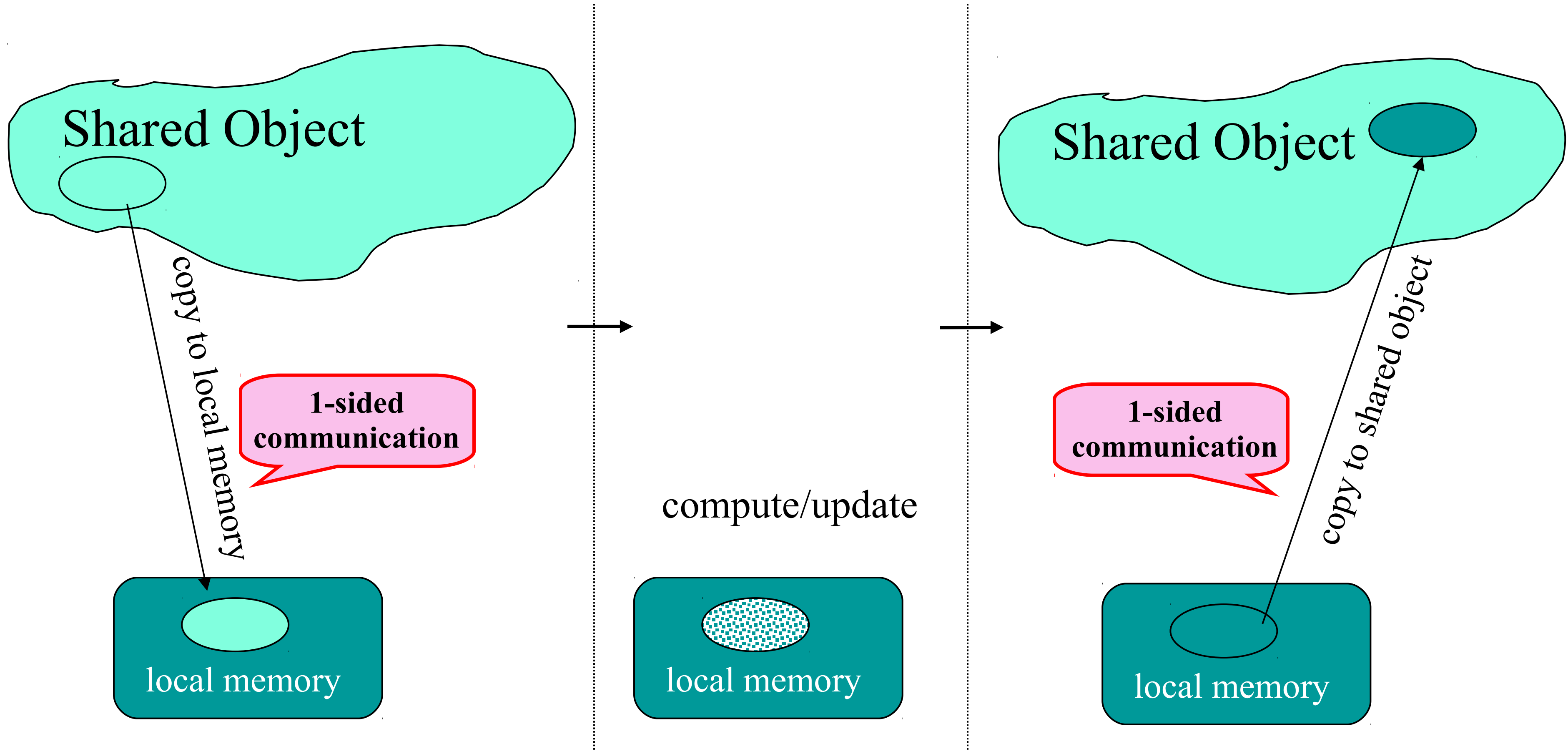
Single, shared data structure

- 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

- A legacy of Jarek Nieplocha, PNNL

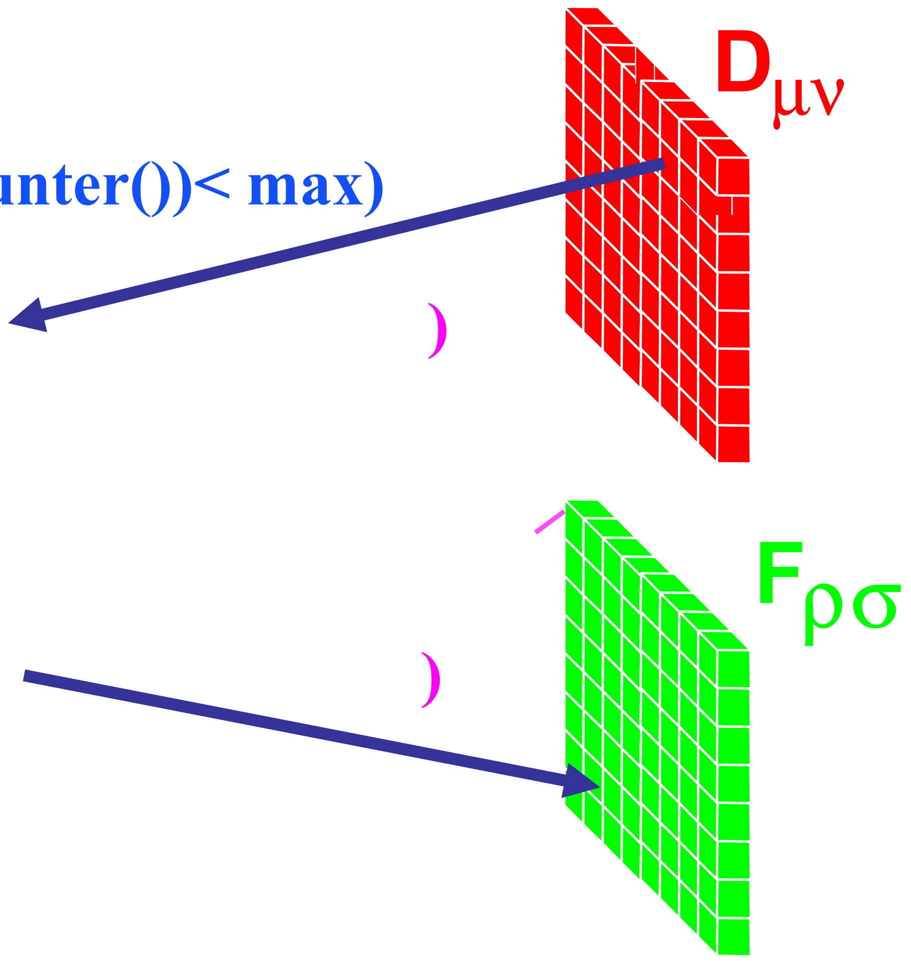
http://www.emsl.pnl.gov/docs/global/

Non-uniform memory access model of computation



Dynamic load balancing

```
While ((task = SharedCounter())< max)
        call ga_get(                    )
              (do work)
        call ga_acc(                    )
End while
Barrier()
```



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, ...

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

B = block size

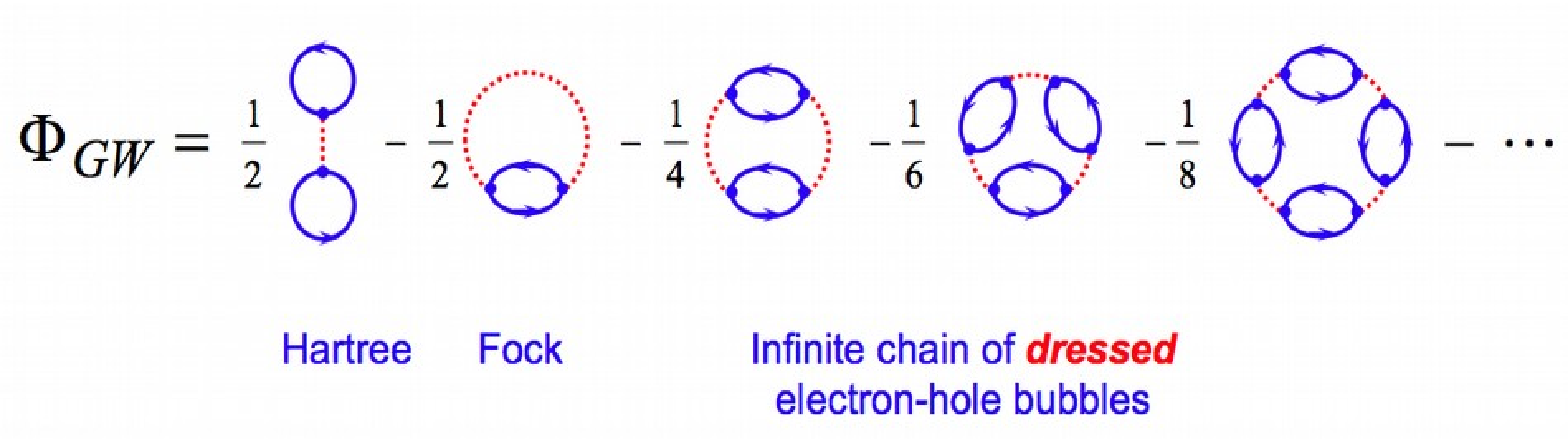
$t_{\mathrm{comm}} = O(B^2)$ $t_{\mathrm{compute}} = O(B^4)$ $\frac{t_{\mathrm{compute}}}{t_{\mathrm{comm}}} = O(B^2)$

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


$\Phi_{GW} = \frac{1}{2}$ $- \frac{1}{2}$ $- \frac{1}{4}$ $- \frac{1}{6}$ $- \frac{1}{8}$ $- \cdots$
Hartree
Fock
Infinite chain of *dressed* electron-hole bubbles

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,
        {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,
        {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
        {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
        {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]
```

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

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/

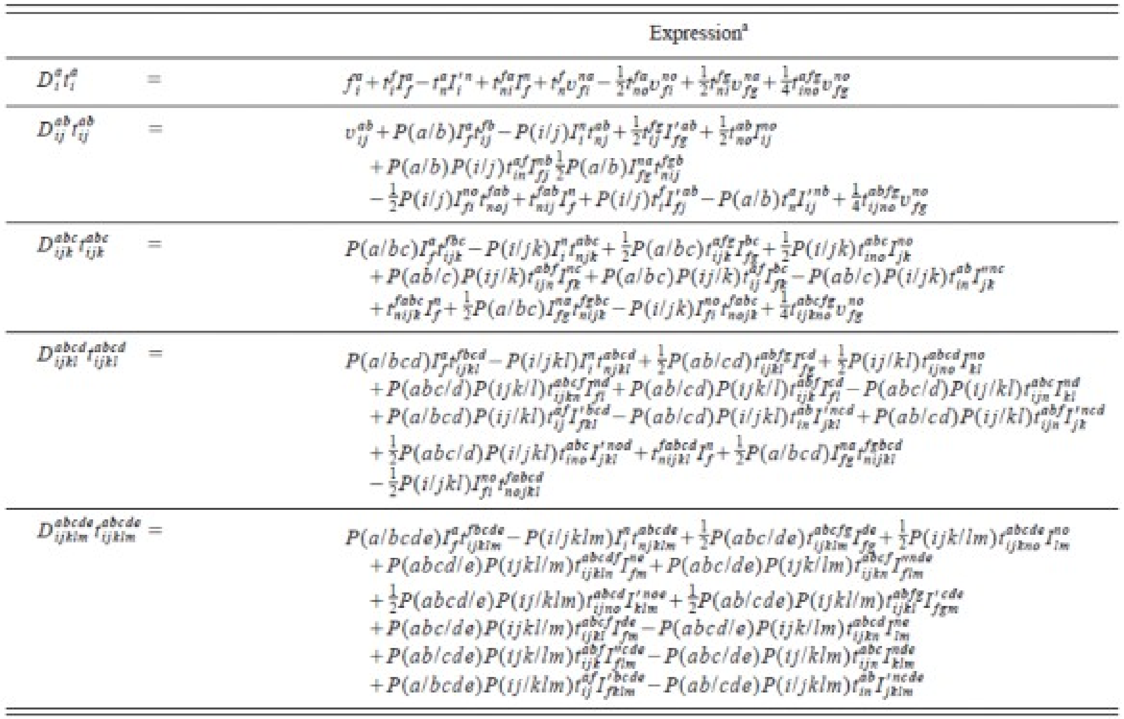
Research at ORNL supported by the Laboratory Directed Research and Development Program. Research at PNNL supported by the Office of Basic Energy Sciences, U. S. Dept. of Energy. Research at OSU, Waterloo, and LSU supported by the National Science Foundation Information Technology Research Program

Tensor Contraction Engine (TCE) (Kowalski, PNNL)

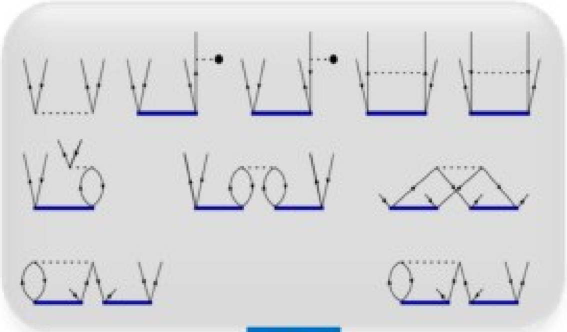
EMSL

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



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

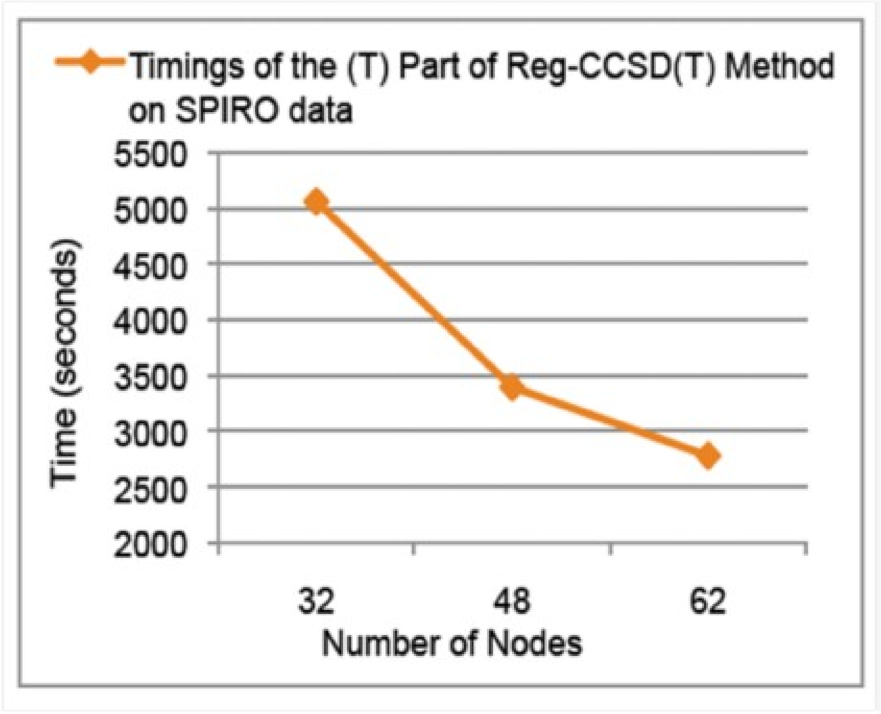
```
next = NXTASK(nprocs, 1)
DO p3b = noab+1,noab+nvab
DO p4b = p3b,noab+nvab
DO h1b = 1,noab
DO h2b = h1b,noab
IF (next.eq.count) THEN
CALL GET_HASH_BLOCK(d_a,dbl_mb(k_a),dim
- 1 + (noab+nvab) * (h1b_1 - 1 + (noab+
+nvab) * (p3b_1 - 1)))))
CALL GET_HASH_BLOCK_I(d_a,dbl_mb(k_a),d
```

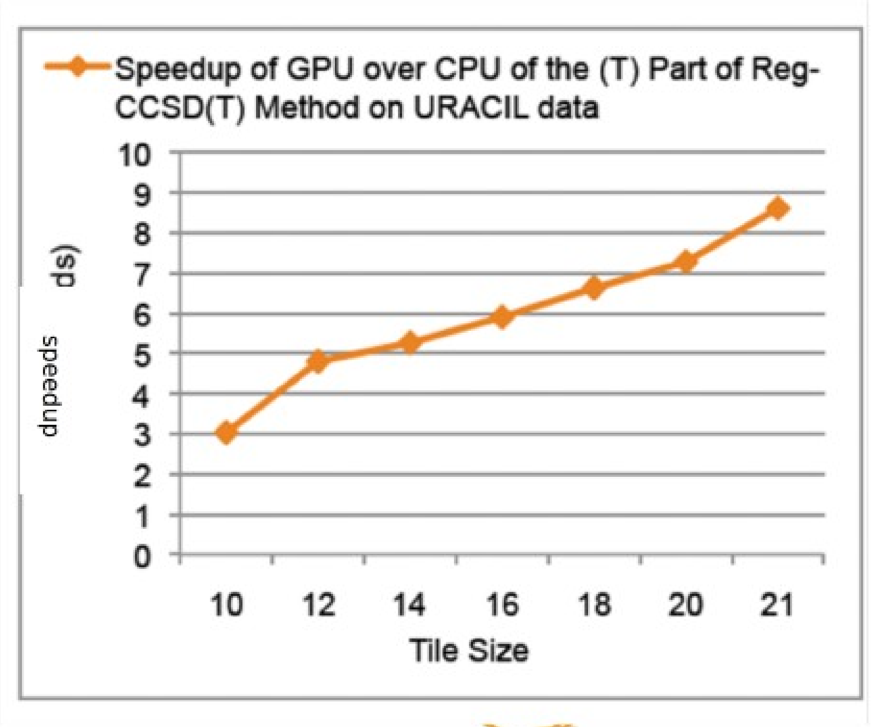
Pacific Northwest
NATIONAL LABORATORY

Office of Science
U.S. DEPARTMENT OF ENERGY

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





MADNESS

狂

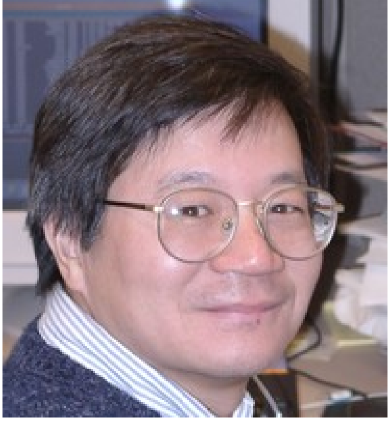
Multiresolution
Adaptive
Numerical
Scientific
Simulation

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


George Fann

Judy Hill

Gregory Beylkin

Rebecca Hartman-Baker

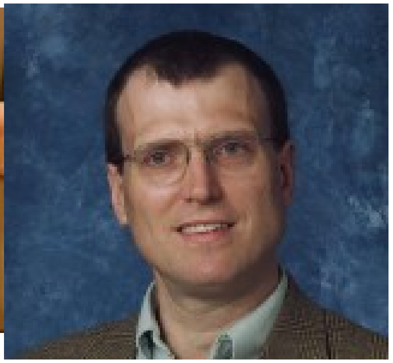
Jeff Hammond

Ariana Beste

Eduard Valeyev

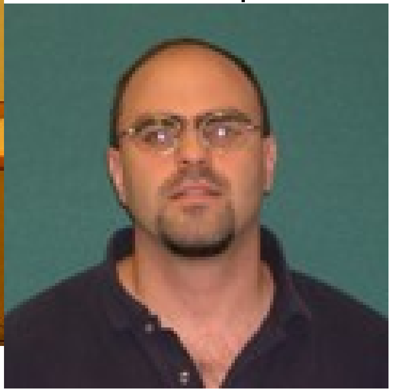
Alvaro Vasquez

Hideo Sekino


Robert Harrison

Nicholas Vence

Takahiro Ii


Scott Thornton

Matt Reuter

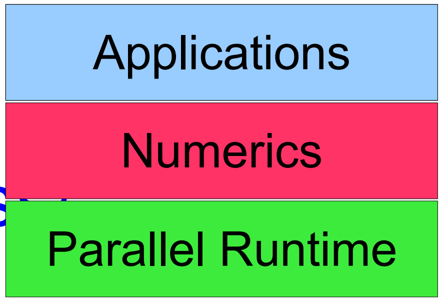
Nichols Romero

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/madnes



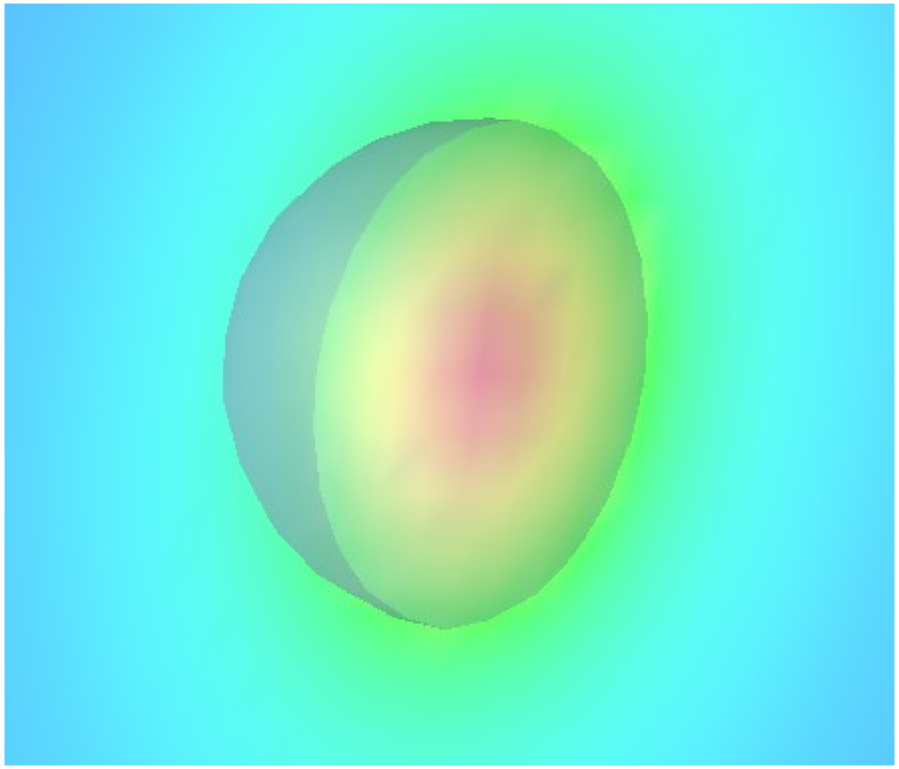
Big picture

- 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

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.

He atom Hartree-Fock

```
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
```

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

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

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

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

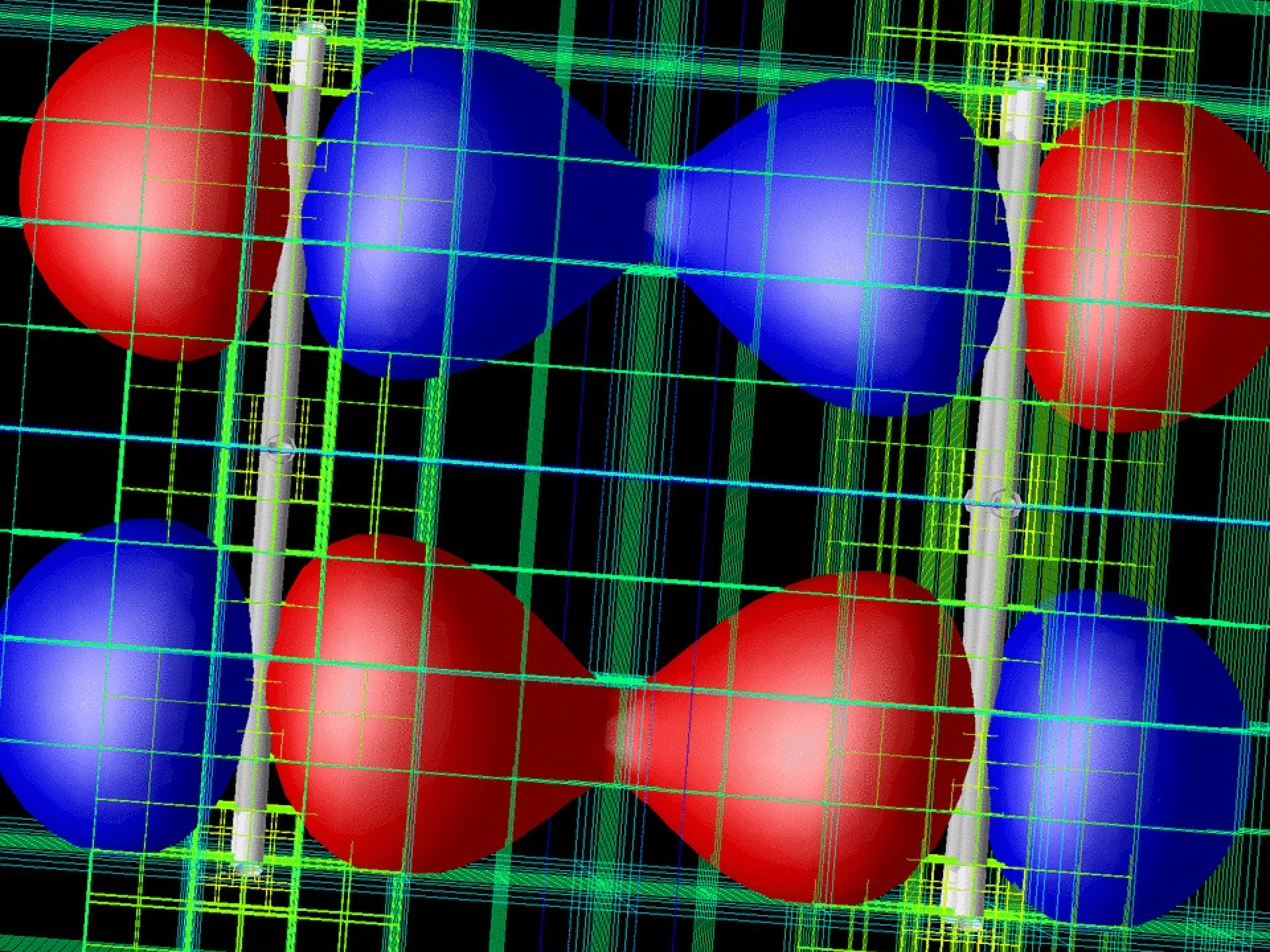
- Low-operator rank

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

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

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



Example tree in Haar basis

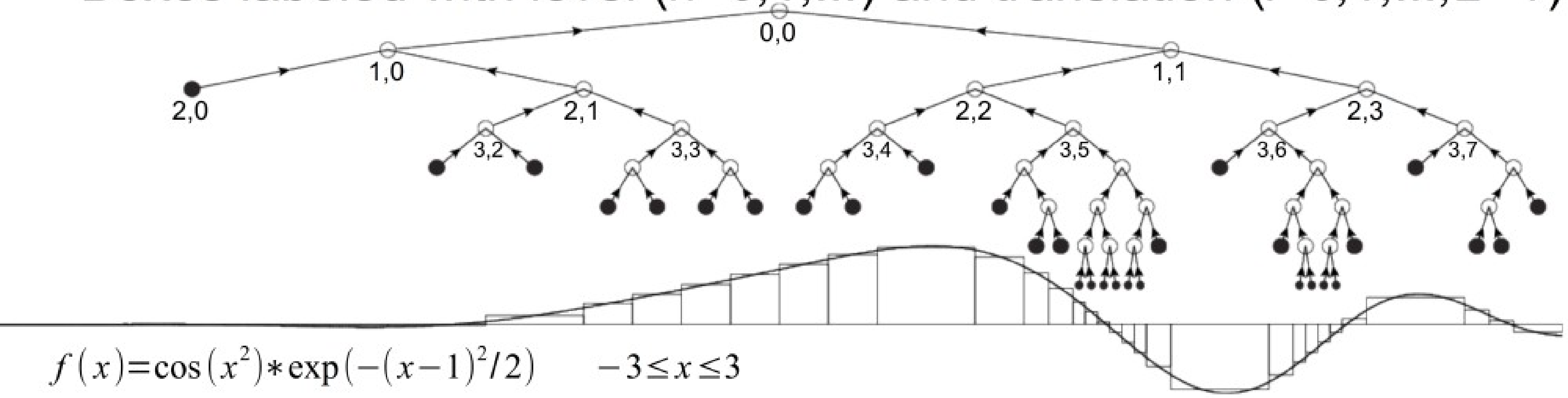
Haar basis is a piecewise constant (like a histogram)
- Not useful for real calculations but easy to visualize and of fundamental importance

Adaptive local refinement until local error measure is satisfied
- Smaller boxes where rate of change is high (and value not negligible)

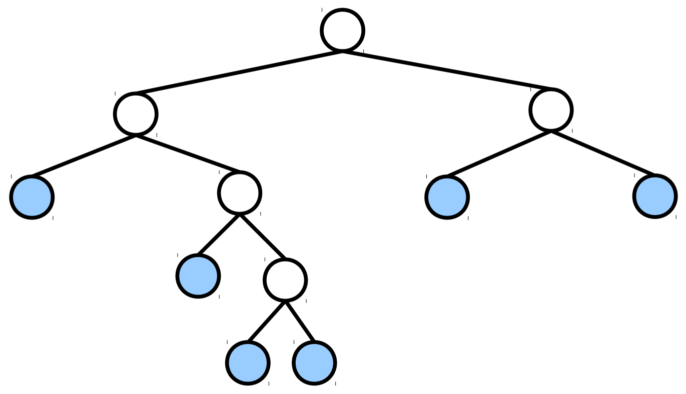
Conventional adaptive mesh corresponds to boxes

Construct tree connecting fine-scale to coarser-scale boxes

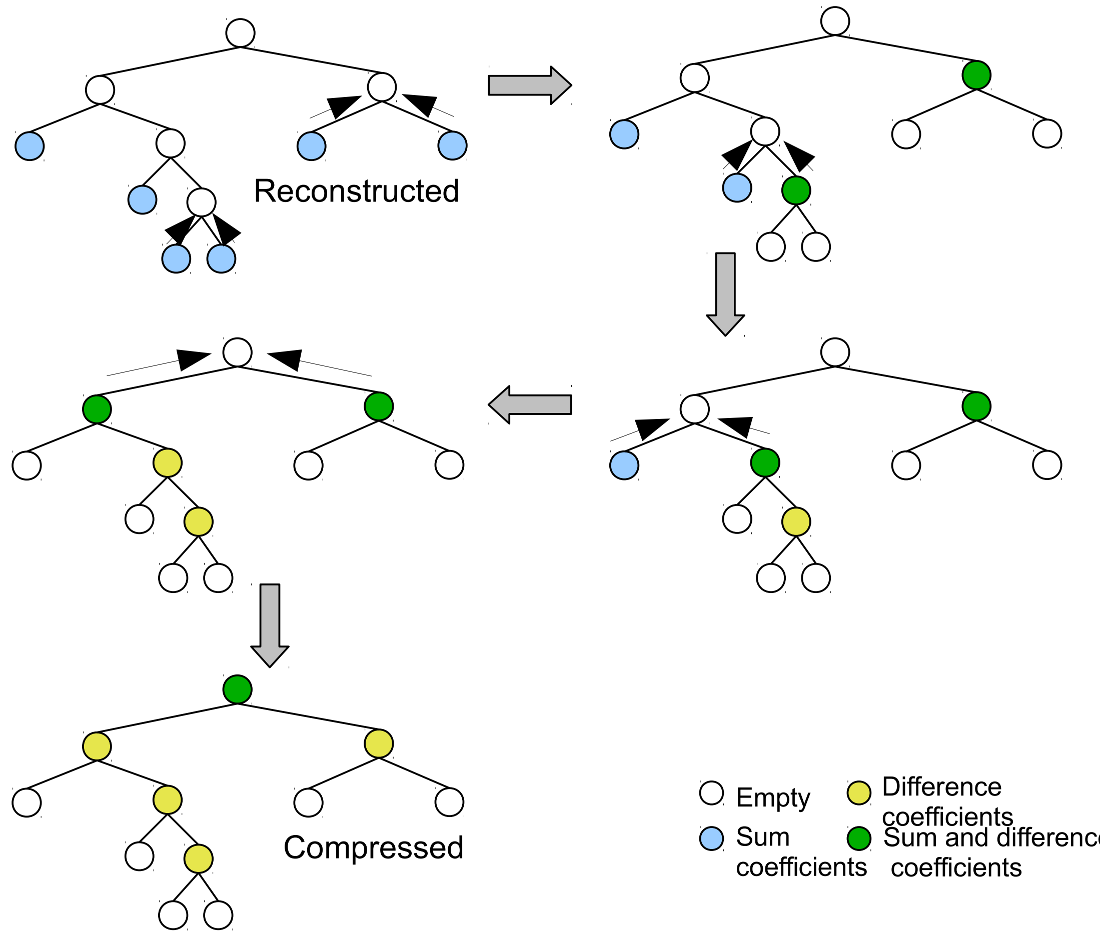
Boxes labeled with level (*n=0,1,...*) and translation (*l=0,1,...,2ⁿ-1*)

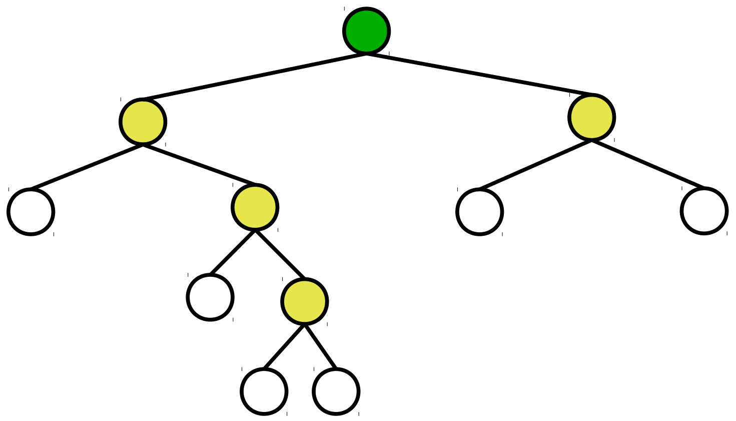


$f(x)=\cos(x^2)*\exp(-(x-1)^2/2) \qquad -3\leq x\leq 3$



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





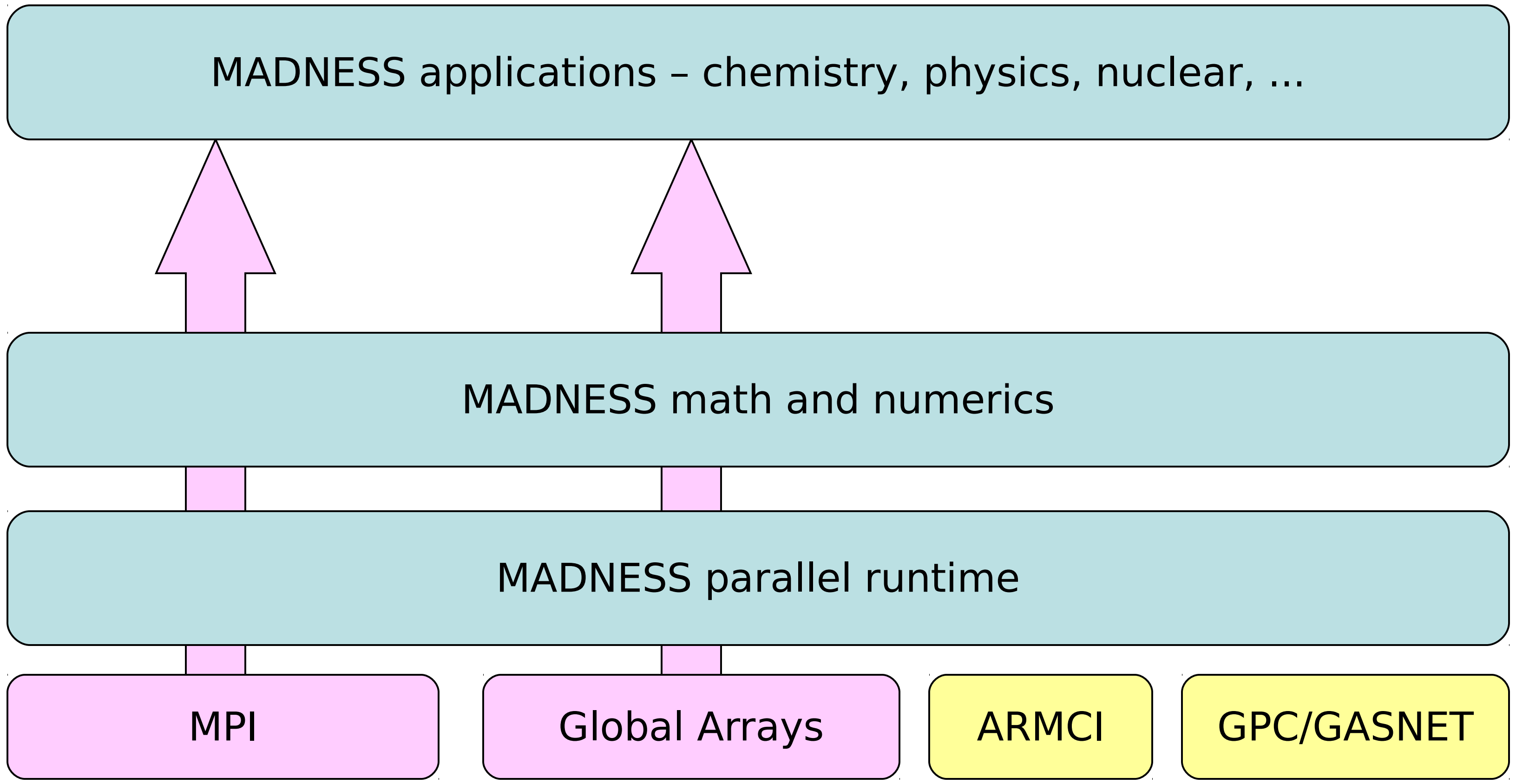
Tree in **compressed** form. Wavelet (difference) coefficients at interior nodes, with scaling functions coefficients also at root. Leaf nodes empty.

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.

MADNESS architecture


MADNESS applications – chemistry, physics, nuclear, ...
MADNESS math and numerics
MADNESS parallel runtime
MPI
Global Arrays
ARMCI
GPC/GASNET

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

- Lowest level: Serialization, active messages
- 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

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?

Serialization

- Convert an object (and containers thereof) into a serial stream of bytes
  - Conceptually based on Boost serialization
  - Symmetric & operator for input/outut
  - ar & x & y & z;
  - Asymmetric operators also (<<, >>)
- Fundamental types and containers
- User types provide supported through both intrusive and non-intrusive methods

Example seralization

```
class A {
     float a;
 public:
     A(float a = 0.0) : a(a) {}

     template <class Archive>
     inline void serialize(Archive& ar)
       {
           ar & a;
       }
 };
```

Active message interface

```
Class AmArg {
public:
   unsigned char* buf() const;
   ProcessID get_src() const;
   World* get_world() const;
   Archive& operator&(const T& t) const;
   Archive& operator&(T& t) const;
};
typedef void (*am_handlerT) (const AmArg&);

void send(ProcessID dest, am_handlerT op,
          const AmArg* arg, int attr);
```

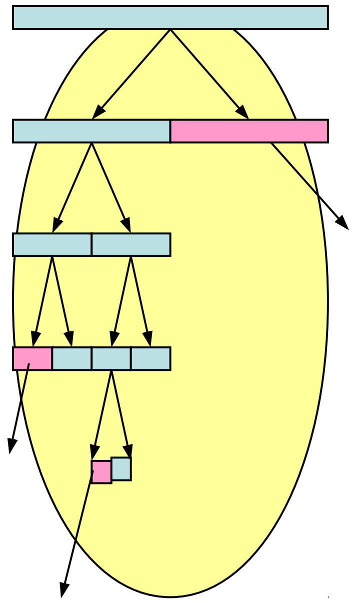
Futures

- 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);
```

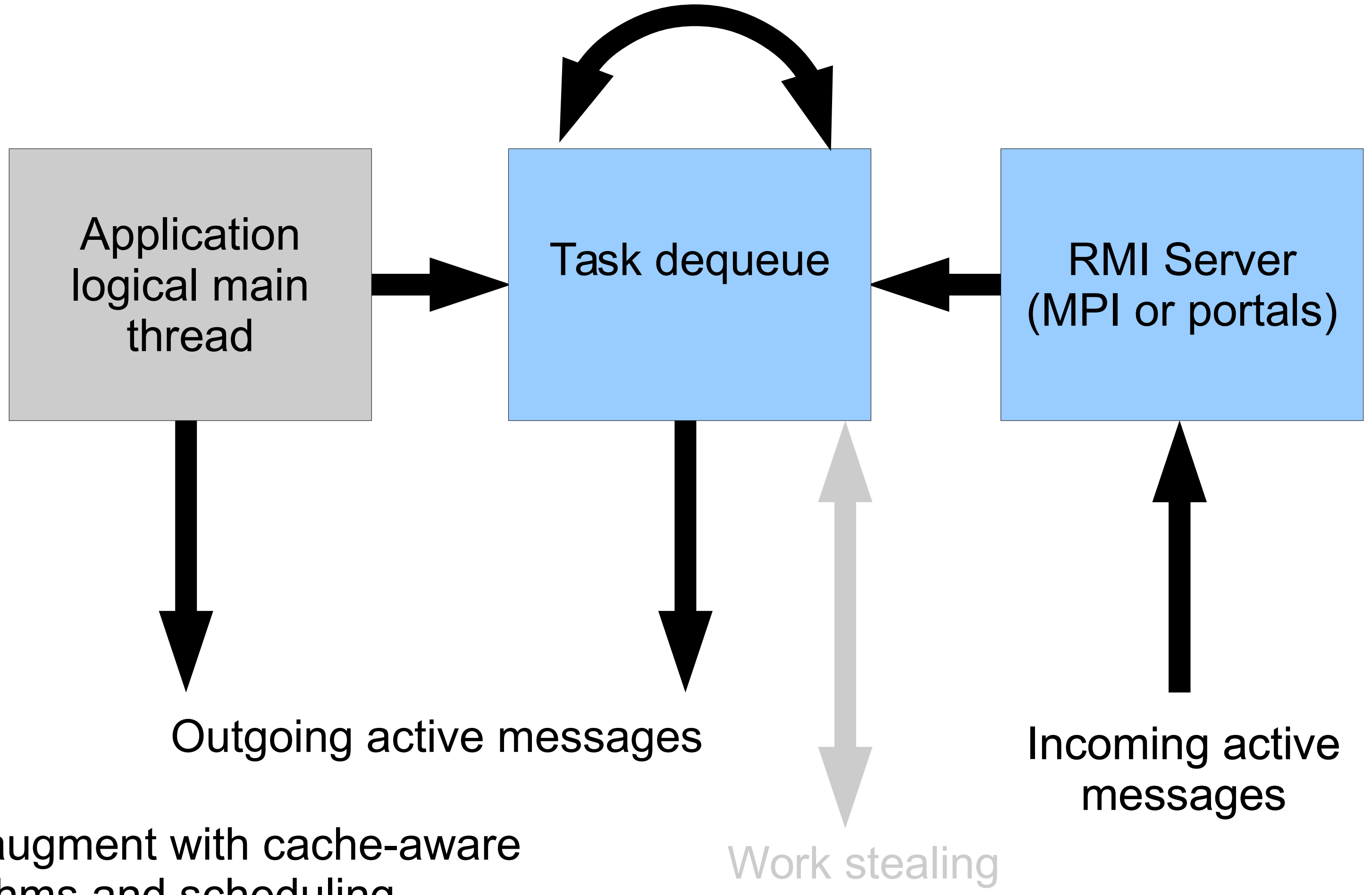
A container is created mapping indices to values.

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

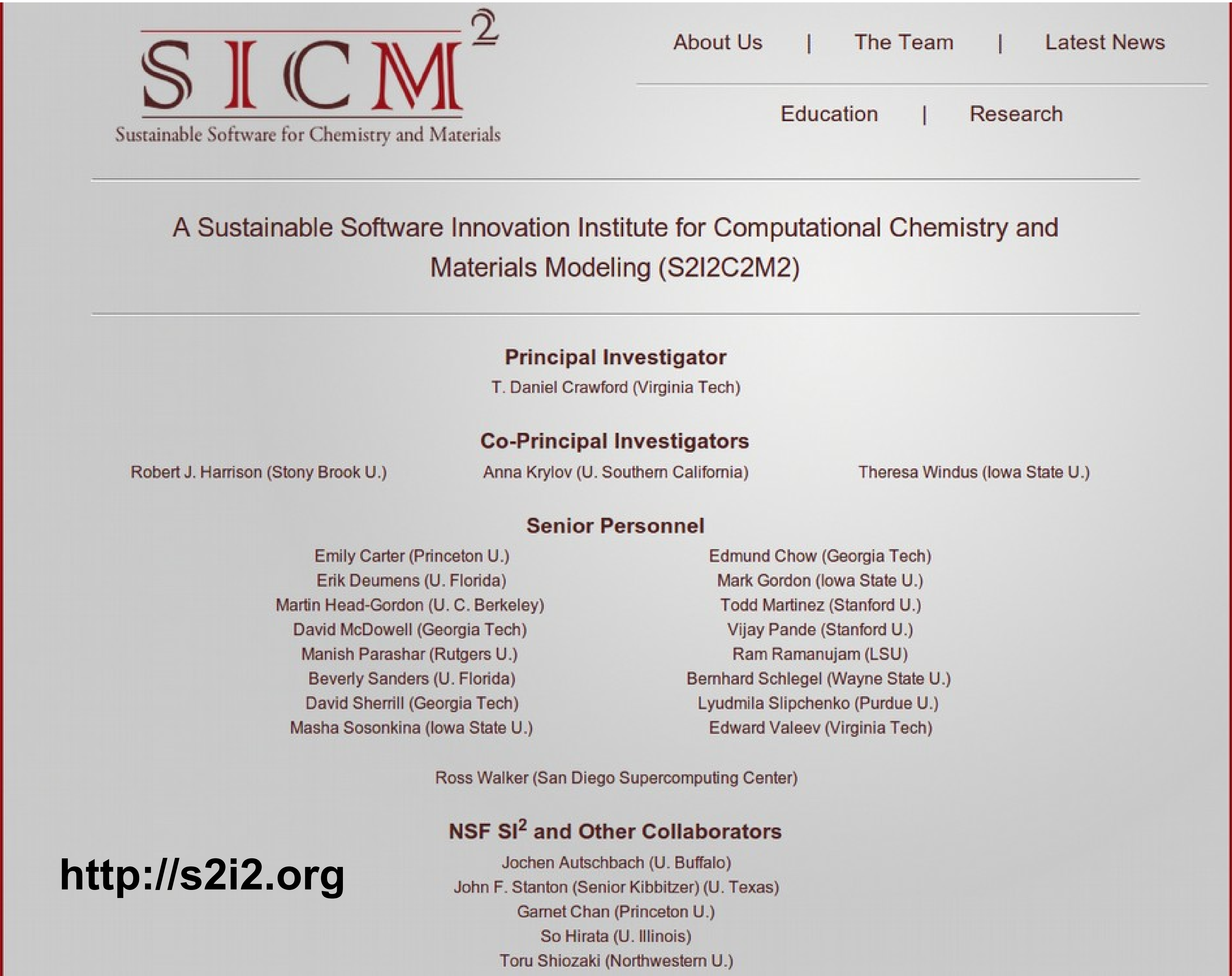
Some issues

- 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.

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


SICM²
Sustainable Software for Chemistry and Materials
About Us | The Team | Latest News
Education | Research

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 multidisciplina

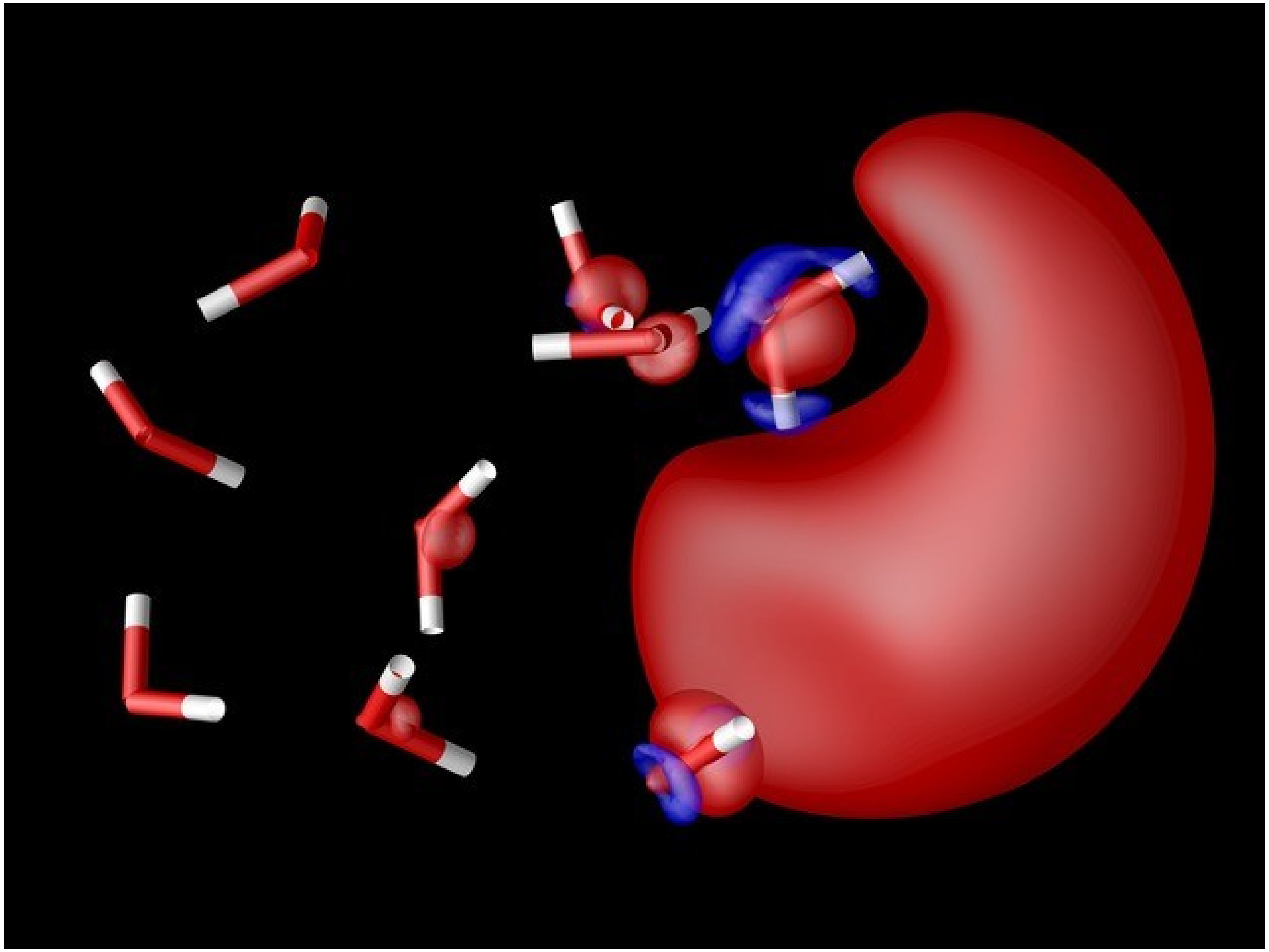
Stony Brook University

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

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

Molecular Electronic Structure



Energy and gradients

ECPs coming (Sekino, Thornton)

Response properties (Vasquez, Yoko Sekino)

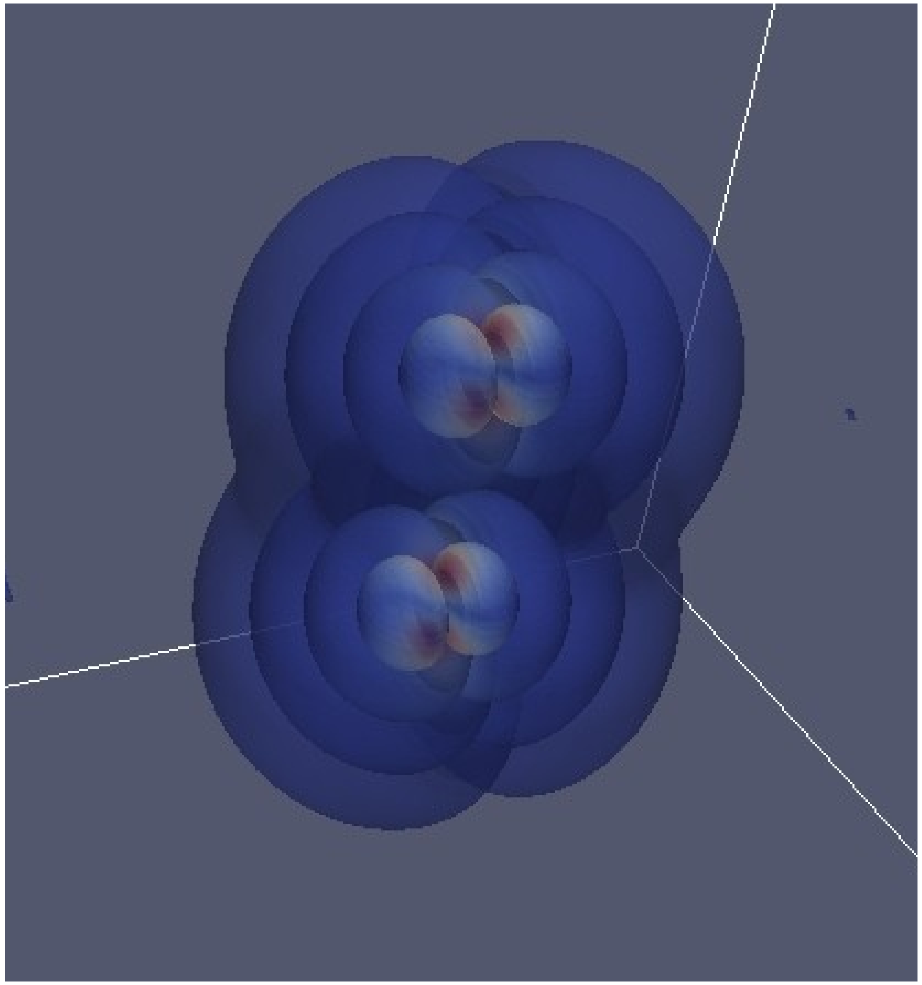
Still not as functional as previous Python version of Yanai

Spin density of solvated electron

Nuclear physics

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

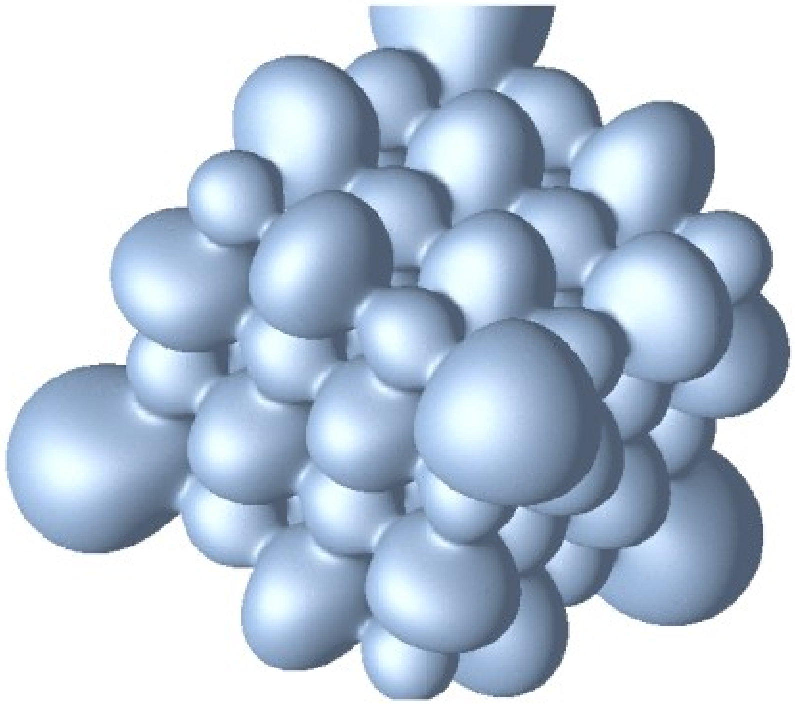
- DOE UNDEF
- Nuclei & neutron matter
- ASLDA
- Hartree-Fock Bogliobulov
- Spinors
- Gamov states

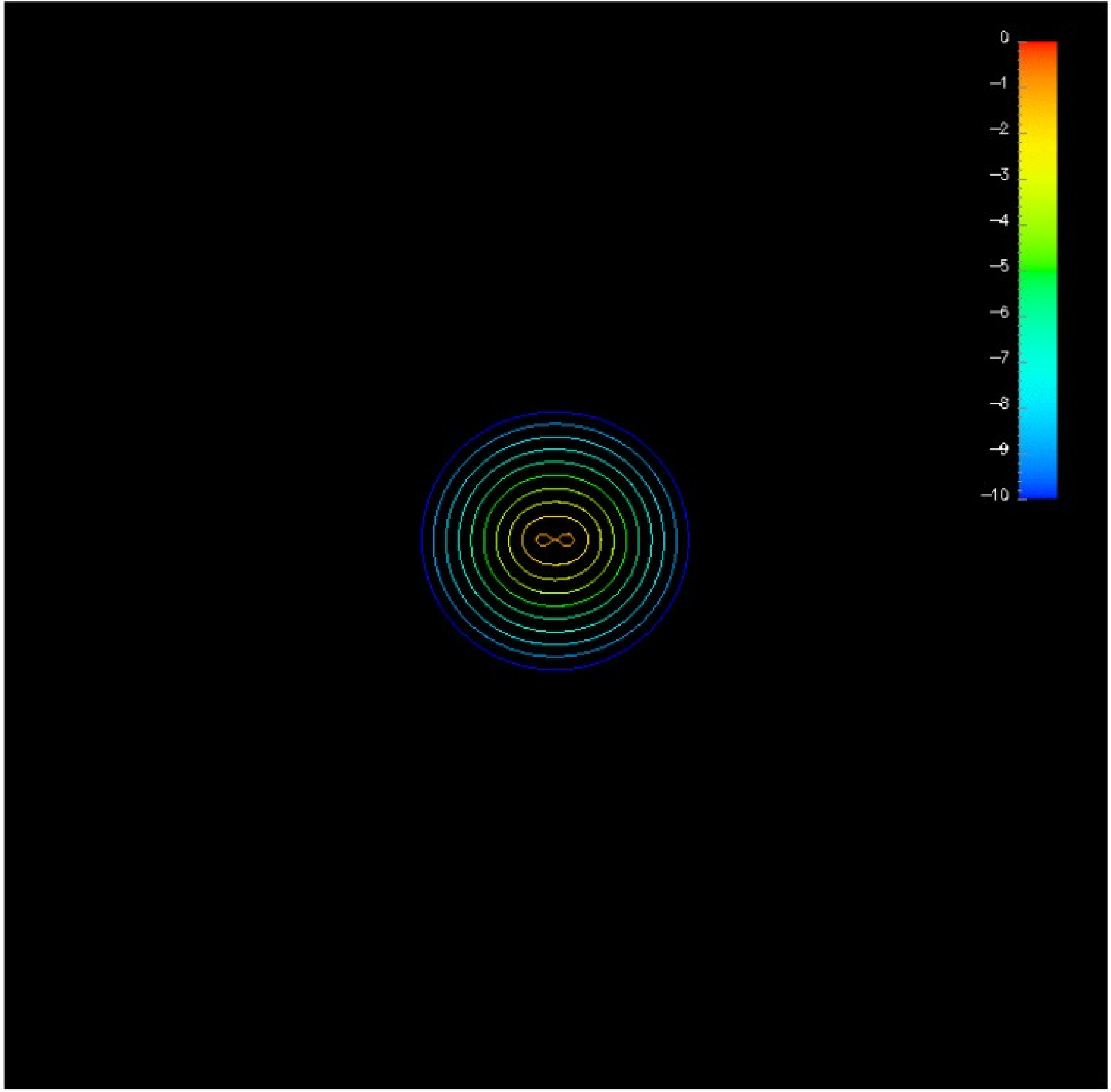


Imaginary part of the seventh eigen function
two-well Wood-Saxon potential

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

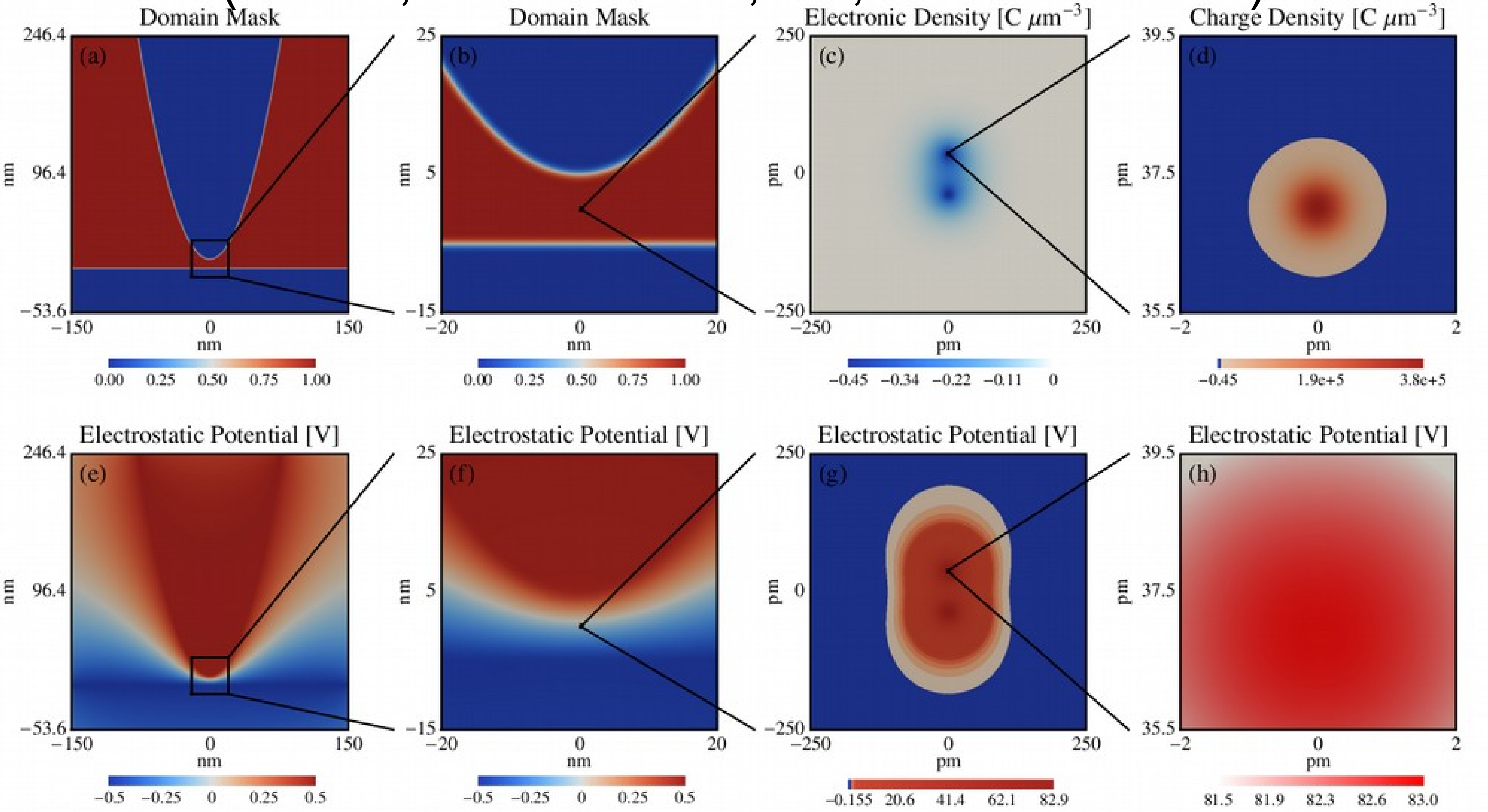


dependent electronic structure

Vence, Krstic, Harrison UT/ORNL

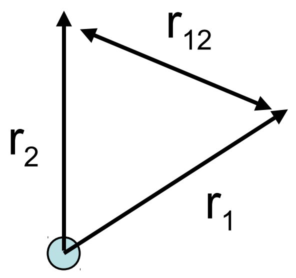
H_2^+ molecule in laser field (fixed nuclei)

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 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

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

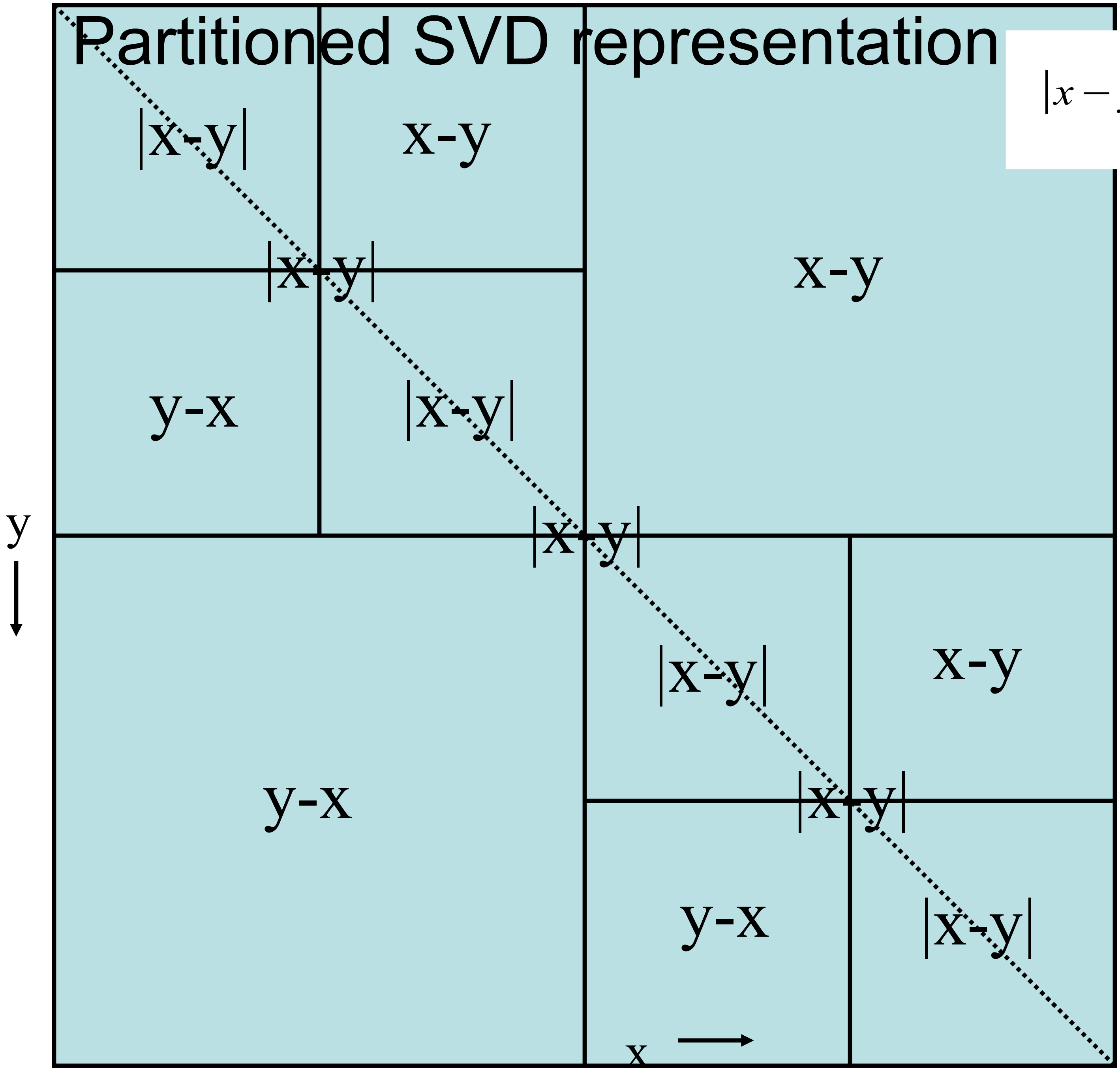
- Include the inter-electron distance in the wavefunction
  - E.g., Hylleraas 1938 wavefunction for He

$$\Psi(r_{1,} r_{2,} r_{12}) = \exp(-\xi(r_1 + r_2))(1 + a\, r_{12} + \cdots)$$

  - 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 ε)*

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.

Can also send an active message

```
#include <world/world.h>
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);
    }

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

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)

Summary

- MADNESS is a general purpose framework for scientific simulation
  - Conceived for the next (not the last) decade
  - Increases HPC productivity by reducing many sources of complexity
  - Deploys advanced math, numerics, and C/S

http://code.google.com/p/m-a-d-n-e-s-s

Funding

- DOE: 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

References

B. Alpert, G. Beylkin, D. Gines, and L. Vozovoi, *"Adaptive Solution of Partial Differential Equations in Multiwavelet Bases,"* Journal of Computational Physics, v. 182, pp. 149-190, 2002

G.I. Fann, G. Beylkin, R.J. Harrison and K.E. Jordan, *"Singular operators in multiwavelet bases,"* IBM J. Res. Dev., 48 (2004) 161.

R.J. Harrison, G.I. Fann, T. Yanai, Z. Gan and G. Beylkin, *"Multiresolution quantum chemistry: basic theory and initial applications,"* J. Chem. Phys., 121 (2004) 11587.

T. Yanai, G.I. Fann, Z. Gan, R.J. Harrison, G. Beylkin, *"Multiresolution quantum chemistry in multiwavelet bases: Analytic derivatives for Hartree-Fock and density functional theory,"* J. Chem. Phys., 121 (2004) 2866.

T. Yanai, R.J. Harrison and N.C. Handy, *"Multiresolution quantum chemistry in multiwavelet bases: time-dependent density functional theory with asymptotically corrected potentials in local density and generalized gradient approximations,"* Mol. Phys., 103 (2004) 403.

R. Harrison, G. Fann, Z. Gan, T. Yanai, S. Sugiki, A. Beste, and G. Beylkin, *"Multiresolution computational chemistry,"* J. Physics, Conference Series, 16, 243-246, 2005.

T. Yanai, R.J. Harrison, G.I. Fann and G. Beylkin, *"Multi-resolution quantum chemistry: linear response for excited states,"* J. Chem. Phys., submitted for publication, 2005.

G. Beylkin, R. Cramer, G. Fann and R. J. Harrison, *"Multiresolution separated representations of singular and weakly singular operators,"* Applied and Computational Harmonic Analysis, 23 (2007) 235-253

H. Sekino, Y. Maeda, T. Yanai, and R.J. Harrison, *"Basis set limit Hartree-Fock and density functional theory response property evaluation by multiresolution multiwavelet basis,"* J. Chem. Phys,129(3):034111 , 2008

G.I. Fann, J.C. Pei, R.J. Harrison ,J. Jia, J.C. Hill, M.J. Ou, W. Nazarewicz, W.A. Shelton and N. Schunck., *"Fast multiresolution methods for density functional theory in nuclear physics,"* Journal of Physics, 180, 012080, 2009

M. Reuter, J. Hill and R.J. Harrison, *"Solving PDEs in Irregular Geometries with Multiresolution Methods I: Embedded Dirichlet Boundary Conditions,"* Computer Physics Communications, submitted March 2010