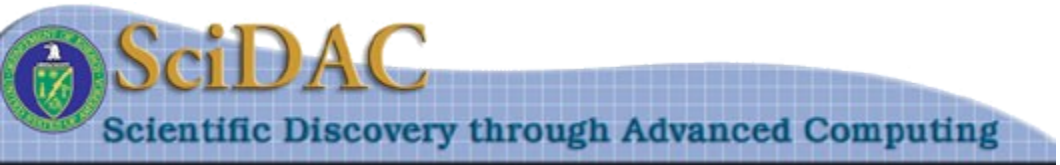
Seeking a sustainable software model for scientific simulation

Robert J. Harrison

harrisonrj@ornl.gov

robert.harrison@utk.edu



Funding

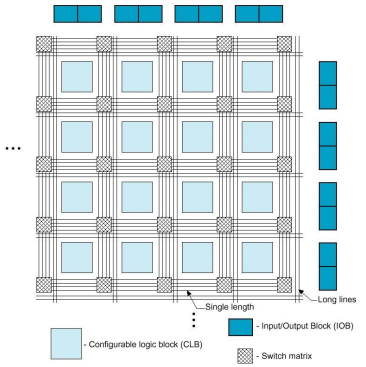
- DOE: Exascale co-deisgn, SciDAC, Office of Science divisions of Advanced Scientific Computing Research and Basic Energy Science, under contract DE-AC05-00OR22725 with Oak Ridge National Laboratory, in part using the National Center for Computational Sciences.
- DARPA HPCS2: HPCS programming language 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

National benefits of exascale and associated technologies

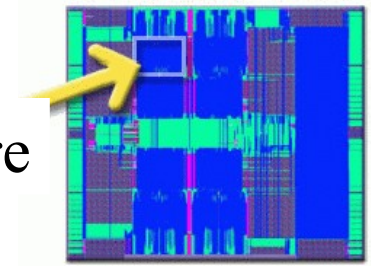
- Basic science currently drives high-end HPC
  - It consumes (nearly) all petascale cycles
  - Product design/engineering at terascale or below
  - Lack of expertise is the major barrier to adoption
- We must change this
  - Mature simulation (e.g., comp. chem.) must eventually become relevant to new technologies, policy decisions, ...
- White house OSTP initiative in HPC (Tom Kalil)
  - Vision of simulation rapidly transferring basic science & engineering knowledge and enabling new technologies

Exascale technologies

- Architecture – data is everything
  - power 0.1 → 100 GFLOP/Watt memory 0.3 → 0.03 byte/FLOP
  - cores 8 → 64-1024+ per node number of cores 100K → 100+M
  - concurrency $10^6 \rightarrow 10^9$
- Will be just a corner of entire ecosystem
  - In 2020 1EF = $100M = 1000 PF
    → 1PF ≤ 0.1M
  - S/W still more expensive than H/W
  - Most science will happen at petascale
- Hardware
  - Will leverage high-end server and professional computing platforms
- Software
  - Must run everywhere



1 core



HPC futures we'd like to avoid

Complexity constrains all of our ambitions (cost & feasibility)

- Science, physics, theory, ...
  - Constantly evolving but can take years to implement
  - Scalable algorithms and math
- Software
  - Crude parallel programming tools with explicit expression & manage-ment of concurrency and data
- Hardware
  - Millions of cores with deep memory hierarchy
  - Power constraints
  - Resiliency

O(1) programmers
O(10,000) nodes
O(100,000) processors
O(100,000,000) threads
and growing

- Growing intrinsic complexity of problem
- Complexity kills … sequential or parallel
  - Expressing concurrency at extreme scale
  - Managing the memory hierarchy
- Semantic gap (Colella)
  - Why are our equations are O(100) lines but the program is O(1M) & growing
  - 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 software

- Why are we not experiencing the same nearly exponential growth in functionality?
  - Level of investment or number of developers?
  - Lack of software interoperability and standards?
  - Competition not cooperation between groups?
  - Shifting scientific objectives?
  - Our problems are intrinsically harder?
  - Failure to embrace/develop higher levels for composing applications?
  - Differing impact of hardware complexity?

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 specifications
  - provides tremendous agility and freedom to explore diverse architectures

Conventional solution

- Problem statement + brain
  → algorithm
- Algorithm + language + brain
  → program
- Compile program

→ executable

- Computer + executable + input
  → result
- The brain is
  - Expensive
  - Finite
  - Not growing exponentially

The only step currently employing HPC in most applications

Image from http://www.ucdmc.ucdavis.edu/welcome/features/20071017_Medicine_whitematter/Photos/head_and_brain.jpg

Cost perspectives

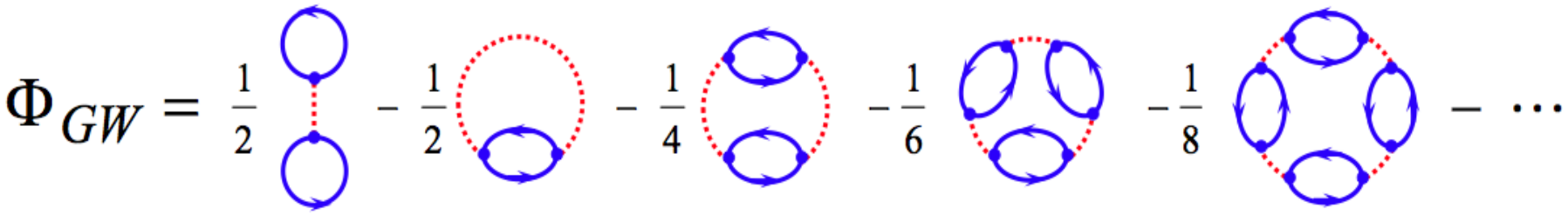
- 250,000 processors running for 12 hours
  - 342 processor years
- Devoting 1+% of runtime resources to load balance and scheduling is quite reasonable
  - 2,500+ processors
- Similarly for transformation, generation, compilation
  - 3.42+ year cpu time
  - What additional transformations are possible?
  - What wall time is acceptable?
  - There is no parallel compiler – “heal thyself?”

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



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,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

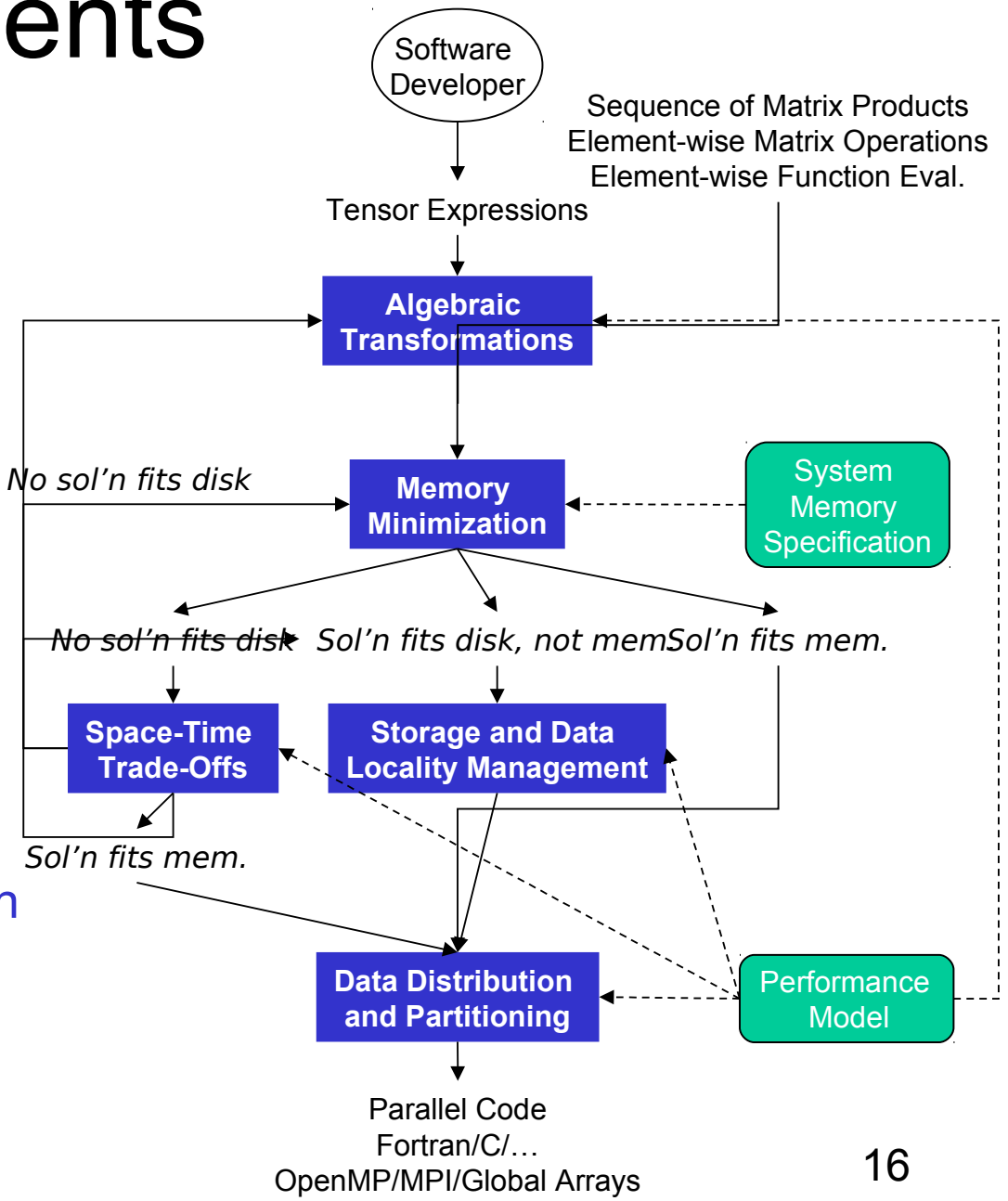
University of Waterloo
Marcel Nooijen, Alexander Auer

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

TCE Components

- Algebraic Transformations
  - Minimize operation count
- Memory Minimization
  - Reduce intermediate storage via loop fusion (LCPC'03)
- Space-Time Transformation
  - Trade-offs between storage and recomputation (PLDI'02)
- Data Locality Optimization
  - Optimize use of storage hierarchy via tiling (ICS'01, HiPC'03, IPDPS'04)
- Data Dist./Comm. Optimization
  - Optimize parallel data layout (IPDPS'03)
- Integrated System
  - (SuperComputing'02, Proc. IEEE 05)



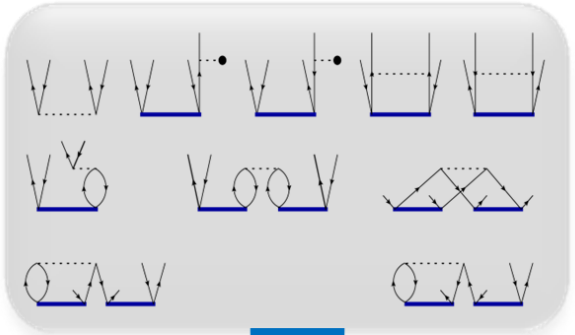
Tensor Contraction Engine (TCE) (Kowalski, PNNL)

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



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

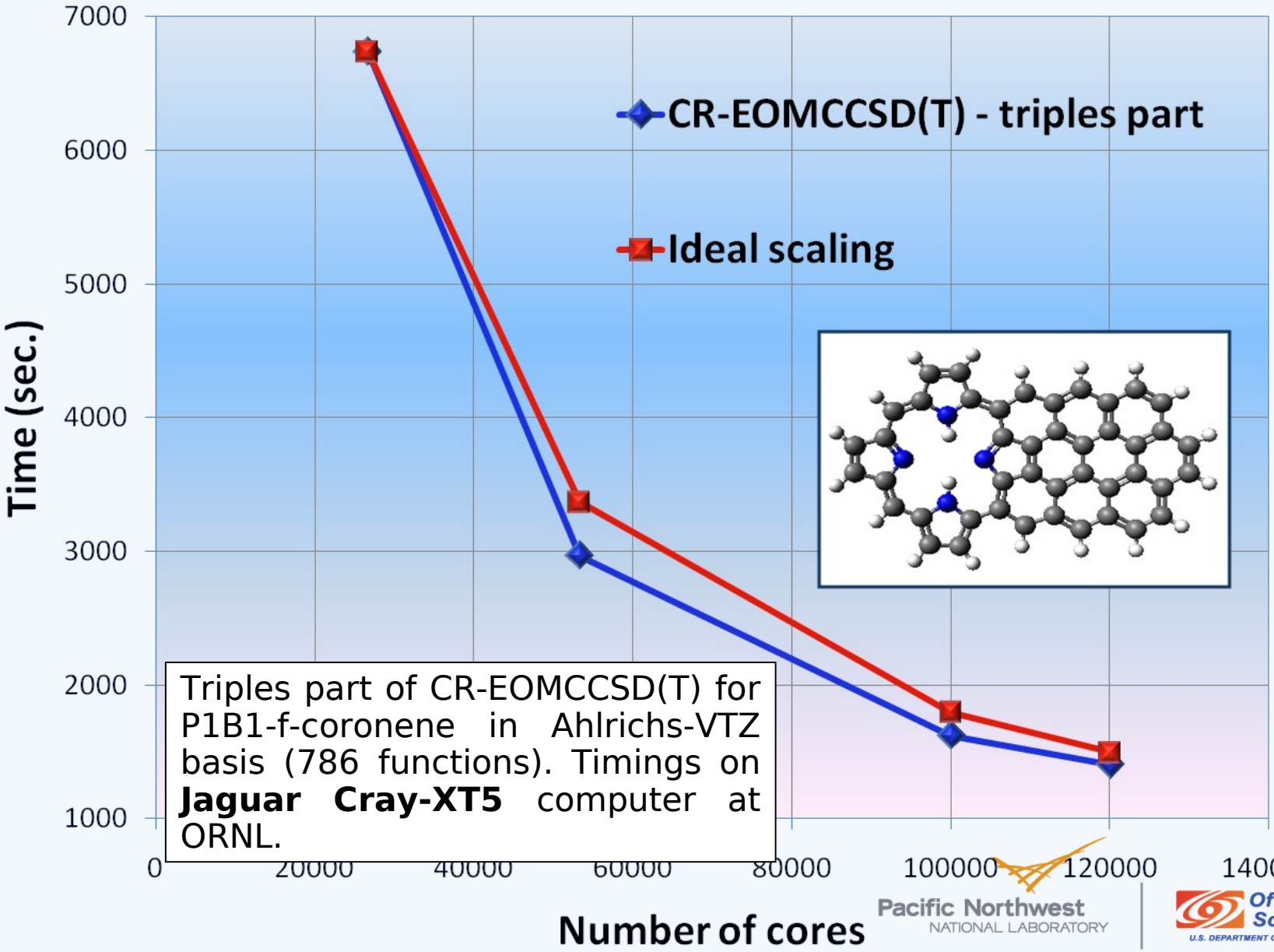
```
next = NXTASK(nprocs, 1)
DO p3b = noab+1,noab+nvab
DO p4b = p3b,noab+nvab
DO h1b = 1,noab
DO h2b = h1b,noab
IF (next.eq.count) THEN
CALL GET_HASH_BLOCK(d_a,dbl_mb(k_a),dim
- 1 + (noab+nvab) * (h1b_1 - 1 + (noab+
+nvab) * (p3b_1 - 1)))))
CALL GET_HASH_BLOCK_I(d_a,dbl_mb(k_a),d
```

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.

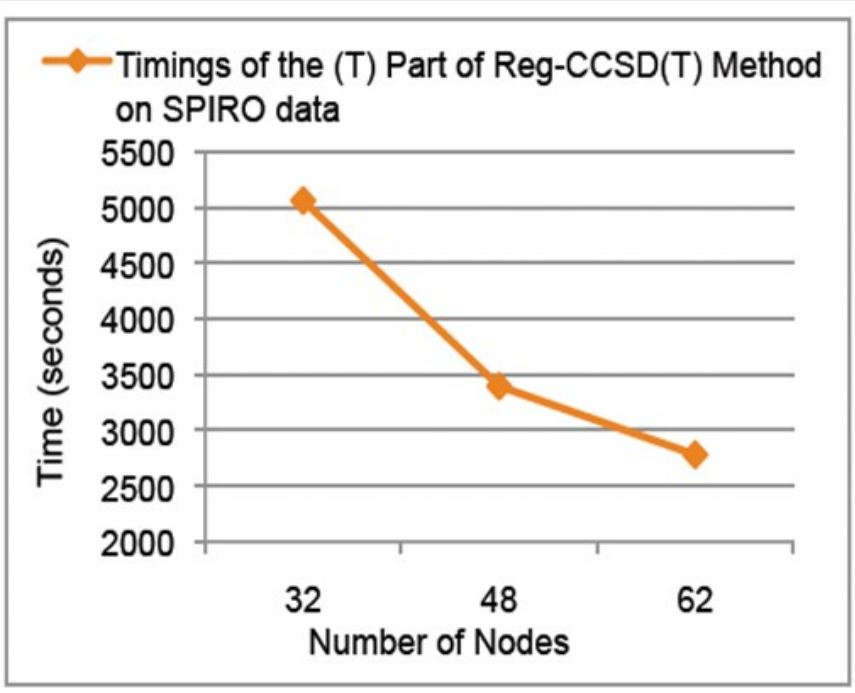
Pacific Northwest NATIONAL LABORATORY

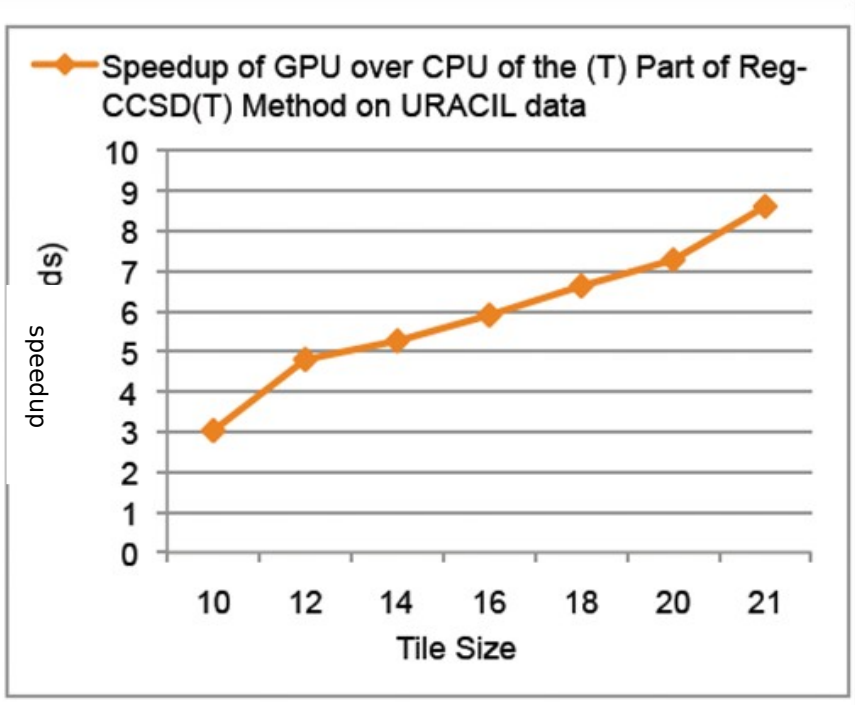
Office of Science U.S. DEPARTMENT OF ENERGY

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

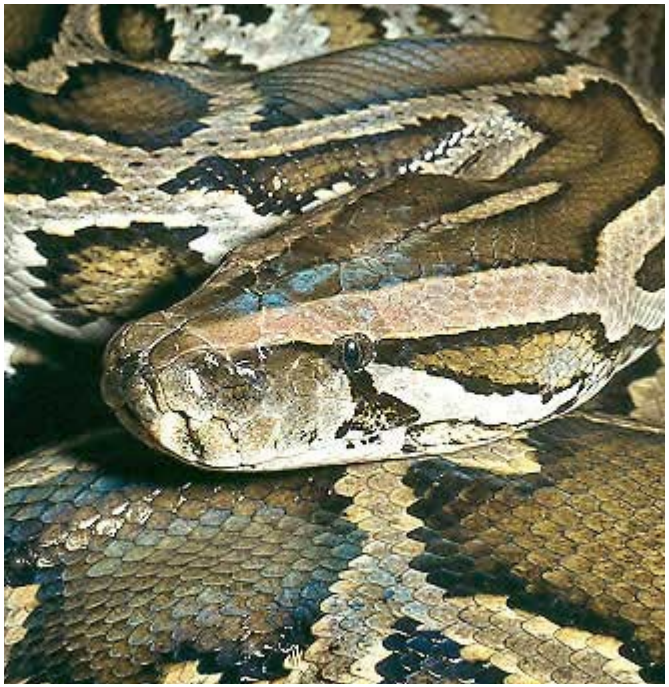
Preliminary tests show very good scalability of the most expensive N7 part of the CCSD(T) approach





Python vs. Java

- The initial Python prototype written by chemists works but has lots of “issues” with memory, speed, ...
- The OSU TCE generated better code, respected bounds on memory use, but was written in Java by C/S graduate students
- And none of the chemists have a clue how it works and none of them know Java
- Guess which is in use



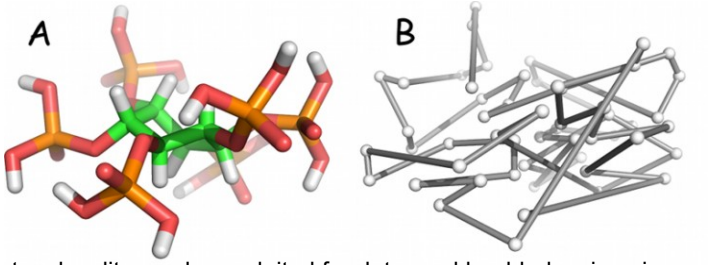
Other challenges for comp. chem.

Robust and power efficient algorithms for one-body Schrodinger – O(105) LOC

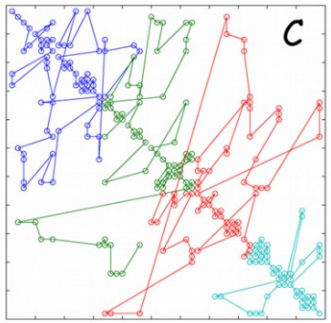
Background: Density functional theory in atomic orbitals, block-sparse trees with fast summation
Science objective: Run at scaling limit for thermodynamic integration of energy-related materials
Issues: Interconnect, power, resilience, scaling, numerical robustness, at scaling limit data motion dominates, irregular and small non-square matrices

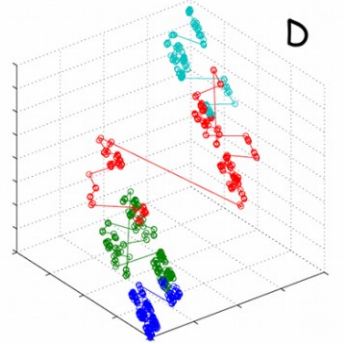
Efficient and resilient algorithms to evaluate two-electron integrals – O(105) LOC

Background: Multiple algorithms – recursion, special functions, quadrature; near min.op. algorithms obtain ~40% peak on x86-64, but no satisfactory solution yet on current accelerators
Science objective: Increased accuracy and speed, more types of bases and integral
Issues: CPU/memory architecture, resilience, power, optimal algorithm hard to find (graph search)



Quantum locality can be exploited for data- and load-balancing via space-filling curves, from atoms (A-B) through matrices (C) to the product space (D).





typical recurrence relation for Coulomb integral (ERI)

$$
\begin{aligned}
(\mathbf{a}+1_i\,\mathbf{b}|\mathbf{cd})^{(m)} &= PA_i(\mathbf{ab}|\mathbf{cd})^{(m)} + WP_i(\mathbf{ab}|\mathbf{cd})^{(m+1)} \\
&+\frac{a_i}{2\zeta}\left[(\mathbf{a}-1_i\,\mathbf{b}|\mathbf{cd})^{(m)} - \frac{\rho}{\zeta}(\mathbf{a}-1_i\,\mathbf{b}|\mathbf{cd})^{(m+1)}\right] \\
&+\frac{b_i}{2\zeta}\left[(\mathbf{a}\,\mathbf{b}-1_i|\mathbf{cd})^{(m)} - \frac{\rho}{\zeta}(\mathbf{a}\,\mathbf{b}-1_i|\mathbf{cd})^{(m+1)}\right] \\
&+\frac{c_i}{2(\zeta+\eta)}(\mathbf{ab}|\mathbf{c}-1_i\,\mathbf{d})^{(m+1)} \\
&+\frac{d_i}{2(\zeta+\eta)}(\mathbf{ab}|\mathbf{c}\,\mathbf{d}-1_i)^{(m+1)}
\end{aligned}
$$

- as many as 20 relations applicable to a given integral
- all different computational complexity
- open-ended graph-search problem
- manually optimized programs are hard to write and maintain as hardware evaloves

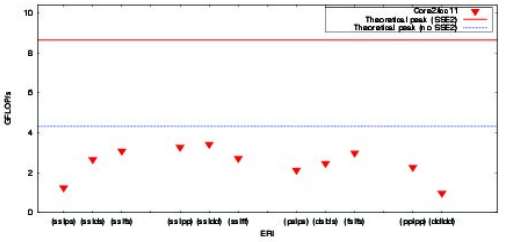
Obara, Saika, *J. Chem. Phys.* 84, 3963(1986).

embedded DSL specification in Libint

```
DGVertex* ABCD_m = new ERIShellQuartet(a,b,c,d,m);
DGVertex* ABCD_mp1 = new ERIShellQuartet(a,b,c,d,m+1);
DGExpression expr = Vector("PA")[i] * ABCD_m +
                    Vector("WP")[i] * ABCD_mp1;
if (a[i]) {
a[i]--;
DGVertex* Am1BCD_m = new ERIShellQuartet(a,b,c,d,m);
DGVertex* Am1BCD_mp1 = new ERIShellQuartet(a,b,c,d,m+1);
 expr += (a[i]+1) * Scalar("one_over_two_eta") *
          (Am1BCD_m -
           Scalar("rho_over_eta") * Am1BCD_mp1
          );
a[i]++;
}

// and so on....
```

performance of Libint code



MADNESS

Multiresolution
Adaptive
Numerical
Scientific
Simulation

Multiresolution Adaptive Numerical Scientific Simulation

Ariana Beste[1], George I. Fann[1], Robert J. Harrison[1,2], Rebecca Hartman-Baker[1], Judy Hill[1], Jun Jia[1],

[1]Oak Ridge National Laboratory

[2]University of Tennessee, Knoxville

in collaboration with

National Science Foundation
WHERE DISCOVERIES BEGIN

Gregory Beylkin[4], Lucas Monzon[4], Martin Mohlenkamp[5], and Hideo Sekino[6]

[4]University of Colorado

[5]Ohio University

[6]Toyohashi Technical University, Japan

harrisonrj@ornl.gov

SciDAC
Scientific Discovery through Advanced Computing

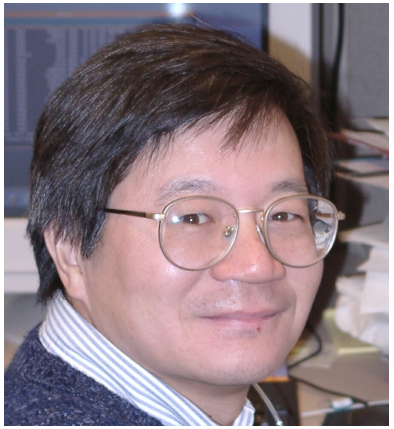
UT-BATTELLE
Oak Ridge National Laboratory

Funding

- MADNESS started as a DOE SciDAC project and the majority of its support still comes from the DOE
- DOE SciDAC, 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

What is MADNESS?

- A general purpose numerical environment for reliable and fast scientific simulation
  - Applications already in nuclear physics, chemistry, atomic physics, material science, with investigations beginning in climate and fusion.
- A general purpose parallel programming environment designed for the petascale
  - Standard C++ with concepts from Cilk, Charm++, HPCS languages, with a multi-threaded runtime that dynamically manages task dependences, scheduling and provides global data view.
  - Compatible by design with existing applications



George Fann

Judy Hill

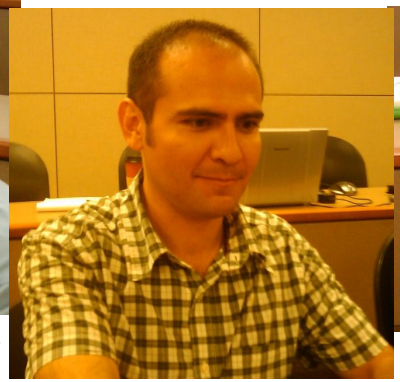
Gregory Beylkin

Rebecca Hartman-Baker

Jun Jia
Tetsuya Kato
Justus Calvin
J. Pei

Ariana Beste

Eduard Valeyev



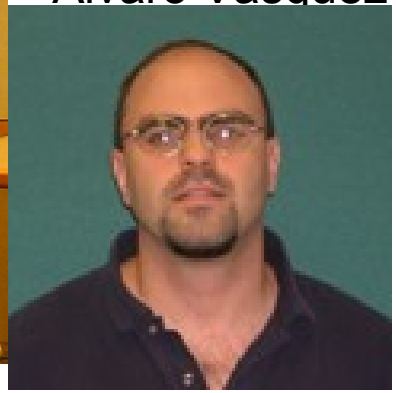
Alvaro Vasquez

Hideo Sekino

Robert Harrison

Nicholas Vence

Takahiro Ii



Scott Thornton

Matt Reuter

Paul Sutter

Why MADNESS

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

The math behind the MADNESS

- Discontinuous spectral element basis
  - High-order convergence ideally suited for modern computer technology
- Multi-resolution analysis for fast algorithms
  - Sparse representation of many integral operators
  - Precision guaranteed through adaptive refinement
- Separated representations of operators and functions
  - Enable efficient computation in many dimensions

Essential techniques for fast computation

- 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

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

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

- Low-operator rank

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

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

Integral Formulation

- Solving the integral equation
  - Eliminates the derivative operator and related “issues”
  - Converges as fixed point iteration *with no preconditioner*

$$\left(-\tfrac{1}{2}\nabla^2 + V\right)\Psi = E\Psi$$

$$\Psi = -2\left(-\nabla^2 - 2E\right)^{-1} V\Psi$$

$$= -2G*\left(V\Psi\right)$$

$$\left(G*f\right)(r) = \int ds \frac{e^{-k|r-s|}}{4\pi|r-s|} f(s) \quad \text{in 3D ; } k^2 = -2E$$

Such Green’s Functions (bound state Helmholtz, Poisson) can be rapidly and accurately applied with a single, sparse matrix vector product.

High-level composition

- Close to the physics

$$E = \langle \psi | -\frac{1}{2}\nabla^2 + V | \psi \rangle + \langle \psi\,\psi | \psi\,\psi \rangle$$

```
operatorT G = CoulombOperator(k, rlo, thresh);
functionT rho = psi*psi;
double twoe = inner(G(rho),rho);
double pe = 2.0*inner(Vnuc*psi,psi);
double ke = 0.0;
for (int axis=0; axis<3; axis++) {
    functionT dpsi = diff(psi,axis);
    ke += inner(dpsi,dpsi);
}
double energy = ke + pe + twoe;
```

Let

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

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

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

$$v = x \to -r(x)^{-1}$$

In

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

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

$$S = \langle \psi | \psi \rangle$$

$$V = \langle \psi | \nu * \psi \rangle$$

$$T = \frac{1}{2} * \sum_{i=0}^{2} (\langle \nabla_i \psi | \nabla_i \psi \rangle)$$

print $S, V, T, \dfrac{T+V}{S}$

End

H atom Energy

H atom actual source

```
Let
  Omega = [-20, 20]^3
  r = x -> sqrt(x_0^2 + x_1^2 + x_2^2)
  g = x -> exp(-r(x))
  v = x -> -r(x)^-1
In
  psi = F g
  nu = F v
  S = < psi | psi >
  V = < psi | nu * psi >
  T = 1/2 * sum_i=0^2 < del_i psi | del_i psi >
  print S, V, T, (T + V)/S
End
```

Let

$$\Omega = [-20, 20]^6$$

$$r1 = x \to \sqrt{x_0^2 + x_1^2 + x_2^2}$$

$$r2 = x \to \sqrt{x_3^2 + x_4^2 + x_5^2}$$

$$r12 = x \to \sqrt{(x_0 - x_3)^2 + (x_1 - x_4)^2 + (x_2 - x_5)^2}$$

$$g = x \to \left(1 + \frac{1}{2} * r12\,(x)\right) * \exp\left(-2 * (r1\,(x) + r2\,(x))\right)$$

$$v = x \to -\frac{2}{r1\,(x)} - \frac{2}{r2\,(x)} + \frac{1}{r12\,(x)}$$

In

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

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

$$S = \langle \psi | \psi \rangle$$

$$V = \langle \psi | \nu * \psi \rangle$$

$$T = \frac{1}{2} * \sum_{i=0}^{5} (\langle \nabla_i \psi | \nabla_i \psi \rangle)$$

print $S, V, T, \dfrac{T+V}{S}$

End

He atom Hylleraas 2-term 6D

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

Hartree-Fock

- What I really wanted to type was

$$\min_{\phi} E[\phi] \quad \text{s.t.} \quad \|\phi\|_2 = 1$$

- But had to
  - Provide E (or rather dE/dφ)
  - Describe inexact-Newton algorithm with stopping criterion
  - Transform to integral representation for efficiency and accuracy
- Can automate some steps, c.f. Maple, Mathematica
  - But properties of computation in the underlying basis are crucial for accuracy and efficiency

Runtime Objectives

- Scalability to 1+M processors ASAP
- Runtime responsible for
  - scheduling and placement, managing data dependencies, hiding latency, and medium to coarse grain concurrency
- Compatible with existing models
  - MPI, Global Arrays
- Borrow successful concepts from Cilk, Charm++, Python
- Anticipating next gen. languages

Key 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
- Dynamic load balancing
  - Data redistribution, work stealing, randomization

Futures

- Result of an asynchronous computation
  - Cilk, Java, HPCLs
- 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.

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.

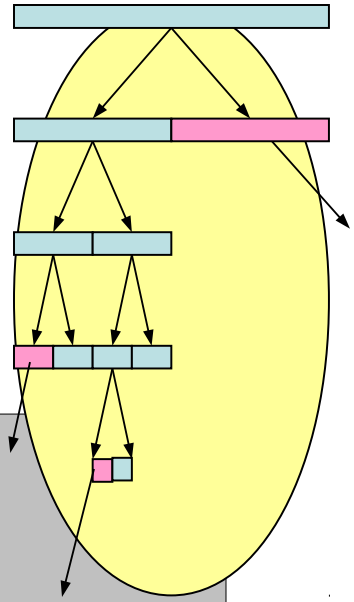
A value is inserted into the container.

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)

Example algorithms

```
compress(node)
    for each child
        coeffs[child]=compress(child)
    return filter(coeffs)
```



```
reconstruct(node,coeffs)
    coeffs=unfilter(coeffs)
    for each child
        reconstruct(child,coeffs[child])
```

```
apply(op,input)
    for each node in input
        for each neighbor
            if norm estimate > tol
                output[neighbor] += compute result
    return output
```

```
multiply(node,left,right)
    if (left & right have coeff) & accurate enuf
        result[node]= left * right
    lc,rc=unfilter(left),unfilter(right)
    for each child
        multiply(child,lc[child],rc[child])
    result[node] = empty
```

```
diff(node,left,right)
    if left & node & right have coeff
        result[node] = stencil ...
    coeffs = unfilter(coeffs)
    for each child
        diff(child,child.left,child.right)
    result[node] = empty
```

Near term objectives

- Separate specification of
  - intent
  - algorithm
  - implementation
- Input form unclear – declarative, imperative, ...
- Generate code for multiple targets
  - Current task-based runtime
  - Map-reduce-like interface (with Cooperman, NEU)
    - aggregation, more amenable to accelerators
- Couple code generation with perf./power model
- Additional coarse grain concurrency
  - More intelligent runtime scheduling and placement

Summary

- We need radical changes in how we compose scientific S/W
  - Complexity at limits of cost and human ability
- DSLs are part of this change
  - Hard part is transformation not translation
  - Need reusable infrastructure and tools
- ~10% of NWChem functionality machine currently machine generated
  - Aiming for at least 60% in about 5 years
  - Don’t know how to do most of this, yet