The Future of
High-Performance
Computational Chemistry
Software

David E. Bernholdt
RJB PhD student 1986-1993

Computer Science and Mathematics
Oak Ridge National Laboratory

bernholdtde@ornl.gov
http://www.csm.ornl.gov/~bernhold/
Software Complexity: The Price of Success?

- As models increase fidelity, software implementing them gets more complex.
- As problems get larger, more complex software is required to implement them efficiently.
- Additional complexity may be introduced (unnecessarily) by developer choices.
- As computers increase in capability, they become more complex.
  - Hardware complexity is exposed to the programmer.
  - Must be managed by the software to obtain the best performance.
- Example:
  - CCSD equations can be expressed in < 100 lines of text.
  - Efficient implementation requires > 10,000 lines of code.
- True of all computational science, not just chemistry.
  - Software complexity impacts overall productivity.
  - Both time to solution & time to first solution.
Dealing with Complexity

• Raise the level of abstraction of the programming model
  – Puts more responsibility on programming model/execution environment
  – Many approaches with different strengths and weaknesses
  – Parallel programming models (libraries)
    • e.g. Global Array model vs. message passing (MPI) for parallel electronic structure codes
  – High-level domain-specific languages
    • e.g. Tensor Contraction Engine

• Develop & use tools to manage remaining complexity
  – Can often help identify and eliminate avoidable complexity too
  – e.g. Scripting languages, makefiles, “standard” libraries, object models, component models
The Common Component Architecture

work by members of the Center for Component Technology for Terascale Simulation Software (ANL, Indiana, LANL, LLNL, ORNL, PNNL, SNL, Utah) and the CCA Forum http://www.cca-forum.org

Research supported by the Mathematics, Information and Computational Sciences Office, Office of Science, U.S. Dept. of Energy.
Basic Concepts of Component-Based Software Engineering (CBSE)

• Component
  – A unit of software deployment/reuse (i.e. has interesting functionality)
  – Interacts with the outside world only through well-defined interfaces
  – Implementation is opaque to the outside world

• Interface (a.k.a. Port in CCA)
  – Defines how components interact, distinct from implementation
  – Generally, a procedural interface
    • Some component-like environments are based strictly on data flow (e.g. AVS, Data Explorer, etc.)
  – Like C++ abst. virtual class, Java interface

• Framework
  – Holds components during application composition and execution
  – Controls the “exchange” of interfaces between components (while ensuring implementations remain hidden)
  – Provides a small set of standard, ubiquitous services to components

• “Plug and play” approach to development of applications
Advantages of CBSE

• CBSE methodology is emerging, especially popular (and successful) in business and internet areas

• Software productivity
  – Provides a “plug and play” application development environment
  – Many components available “off the shelf”
  – Abstract interfaces facilitate reuse and interoperability of software

• “The best software is code you don’t have to write” [Jobs]

• Software complexity
  – Components encapsulate much complexity into “black boxes”
  – Plug and play approach simplifies applications & adaptation
  – Model coupling is natural in component-based approach

• Software performance (indirect)
  – Plug and play approach and rich “off the shelf” component library simplify changes to accommodate different platforms
The Common Component Architecture (CCA)

- CBSE has been developed and is now widespread primarily in non-technical areas
- CBSE has not yet had much uptake in high-performance scientific computing
  - Largely due to deficiencies of “commodity” component models for HPC
- The Common Component Architecture is tailored specifically to the needs of the high-performance scientific computing community
- Supports both parallel and distributed computing
- Designed to be implementable with minimal performance impact
- Minimalist approach makes it easier to incorporate existing code into CCA
- Provides language interoperability for important languages for HPC (Fortran77/90/95, C, C++, Python, Java)
CCA Performance

- Calls between components are equivalent to C++ virtual function calls
  - $O(50\text{ns})$ per call on a 500 MHz Pentium
- Translation of data between languages may add overhead
  - Can be avoided for most scientific software
- Calls within components have no CCA-imposed overhead
- Parallel programming has no CCA-imposed overhead
- Advice: be aware of costs and take them into account in design
  - In practice, overheads are negligible

Overhead of CCA component vs. “native” C++ implementations of a parallel Lennard-Jones molecular dynamics simulation
CCA Application Areas & Component Infrastructure

- Combustion
- Global Climate Modeling
- Quantum Chemistry
- Fusion
- Materials Science & Nanoscience
- Underground radionuclide transport
- Scientific Data Management
- Large-Scale Visualization
- Biomedical Engineering
- Data collection and processing (sensors)

- Distributed Arrays and basic parallel linear algebra
- Parallel Data Redistribution
- Meshing and discretization
- PDE Solvers
- ODE Integrators
- Optimization
- Structured Adaptive Mesh Refinement
- Performance Observation
Component-Based Integration of Chemistry and Optimization Packages: Molecular Geometry Optimization

Yuri Alexeev, Manoj Krishnan, Jarek Nieplocha, Theresa Windus (PNNL)
Curtis Janssen, Joseph Kenny (SNL)
Steve Benson, Lois McInnes, Jason Sarich (ANL)
David Bernholdt (ORNL)
Project Goals

• Move from “proof of concept” stage toward real component-based end-user applications

• Performance evaluation of optimization components
  – Examine efficiency of algorithms in TAO for quantum chemistry

• Further development of optimization capabilities
  – Provide internal coordinate generation, constrained optimization, configurable convergence control

• Graphical user interface to assemble and run applications
  – Provide user-friendly front-end visualization

• Future plans: Exploring chemistry package integration through hybrid calculation schemes and sharing of lower-level intermediates such as integrals and wavefunctions
Software Architecture and Underlying Packages

Quantum Chemistry
- NWChem (PNNL)
- MPQC (SNL)

Optimization
- Toolkit for Advanced Optimization (TAO, ANL)

Linear Algebra
- Global Arrays (PNNL)
- PETSc (ANL)
Benchmarking Optimization Methods

• Compare TAO’s limited memory variable metric (LMVM) method to traditional chemistry methods (i.e. BFGS)

• BFGS updates approximate Hessian at each step using current correction vector pair
  – Quadratic in number of variables for both operation count and memory usage

• LMVM uses guess Hessian and multiple correction pairs (up to 20 in these experiments)
  – LMVM is linear in number of variables for both operation count and memory usage
### Optimization Benchmark Results

*Number of energy/gradient evaluations required with various approaches to converge the structure of four different molecules at the HF/6-31G level from a HF/STO-3G starting point.*

<table>
<thead>
<tr>
<th>Electronic Structure Package</th>
<th>NWChem</th>
<th>NWChem</th>
<th>NWChem</th>
<th>MPQC</th>
<th>MPQC</th>
<th>MPQC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer</td>
<td>NWChem</td>
<td>NWChem</td>
<td>TAO</td>
<td>MPQC</td>
<td>MPQC</td>
<td>TAO</td>
</tr>
<tr>
<td></td>
<td>BFGS</td>
<td>BFGS</td>
<td>LMVM</td>
<td>BFGS</td>
<td>BFGS</td>
<td>LMVM</td>
</tr>
<tr>
<td></td>
<td>No line search</td>
<td>Line search</td>
<td>No line search</td>
<td>No Line search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coordinate System</td>
<td>Cartesian</td>
<td>Cartesian</td>
<td>Cartesian</td>
<td>Cartesian</td>
<td>Cartesian</td>
<td>Cartesian</td>
</tr>
<tr>
<td>Initial Guess Hessian</td>
<td>Diagonal</td>
<td>Diagonal</td>
<td>Diagonal</td>
<td>Transformed internal</td>
<td>Diagonal</td>
<td>Diagonal</td>
</tr>
<tr>
<td>Glycine (10 atoms)</td>
<td>33/33</td>
<td>65/33</td>
<td>19/19</td>
<td>17/17</td>
<td>26/26</td>
<td>19/19</td>
</tr>
<tr>
<td>Isoprene (15)</td>
<td>56/56</td>
<td>89/45</td>
<td>45/45</td>
<td>18/18</td>
<td>75/75</td>
<td>43/43</td>
</tr>
<tr>
<td>Phosphoserine (19)</td>
<td>79/79</td>
<td>121/61</td>
<td>85/85</td>
<td>45/45</td>
<td>85/85</td>
<td>62/62</td>
</tr>
<tr>
<td>Acetylsalicylic acid (21)</td>
<td>43/43</td>
<td>83/42</td>
<td>51/51</td>
<td>24/24</td>
<td>54/54</td>
<td>48/48</td>
</tr>
<tr>
<td>Cholesterol (74)</td>
<td>33/33</td>
<td>&gt;194/&gt;98</td>
<td>30/30</td>
<td>25/25</td>
<td>27/27</td>
<td>30/30</td>
</tr>
</tbody>
</table>
Conclusions on Optimization and Component for Chemistry

Optimization

• LMVM compares well with BFGS, often better
• Having a good initial guess Hessian outweighs LMVM/BFGS differences
  • Diagonal matrix (used in most calculations) is not a good guess Hessian
  • Impact greater on LMVM because it is not a Hessian update method
• Future plans include
  • Allow LMVM to accept guess Hessian provided by chemistry model
  • Expand to larger problems
  • Benchmark other optimization methods available in TAO

Chemistry

• Demonstrated benefits of component approach in chemistry applications
  • Interoperability of chemistry, linear algebra packages
  • Ability to easily utilize software written by experts in other areas (TAO, GA, PETSc)
• Future plans include
  • Hybrid computational schemes, integrating multiple packages
  • Deeper levels of interoperability (integrals, wavefunctions, etc)
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.
The Tensor Contraction Engine Addresses Programming Challenges

- User describes computational problem (tensor contractions, a la many-body methods) in a simple, high-level language
  - Similar to what might be written in papers
- Compiler-like tools translate high-level language into traditional Fortran (or C, or...) code
- Generated code is compiled and linked to libraries providing computational infrastructure

- **Productivity**
  - User writes simple, high-level code
  - Code generation tools do the tedious work
- **Complexity**
  - Significantly reduces complexity visible to programmer
- **Performance**
  - Perform optimizations prior to code generation
  - Automate many decisions humans make empirically
  - Tailor generated code to target computer
  - Tailor generated code to specific problem
So What’s New About This Project?

• The creation of “little languages” and code generation tools has a long history in chemistry and other domains

• Usually viewed only as productivity tools
  – Imitate what researcher would do – but quicker

• We treat it as a computer science problem
  – Similar to (not identical to) an optimizing compiler
  – Algorithmic choices are explored and evaluated rigorously and (in most cases) exhaustively
  – Make use of machine architecture & performance models to specialize generated code to target system

• Target applications
  – Rapid experimentation with new many-body methods
  – Implementation of high-complexity methods
  – Improving computational efficiency on parallel machines
  – Also for nuclear physics…
# Current TCE Capabilities

<table>
<thead>
<tr>
<th>Capability</th>
<th>Prototype TCE</th>
<th>Optimizing TCE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic sequential code generation</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>for CC-based methods</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>QC Packages Interfaced:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• File based</td>
<td>NWChem, UTChem</td>
<td>NWChem</td>
</tr>
<tr>
<td>• General (file, memory, direct)</td>
<td></td>
<td>Under development</td>
</tr>
<tr>
<td><strong>Symmetry Support:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Spin</td>
<td>Spin orbitals</td>
<td>General, in progress</td>
</tr>
<tr>
<td>• Spatial</td>
<td>Abelian</td>
<td>Abelian, in progress</td>
</tr>
<tr>
<td>• Permutational</td>
<td>Fermions</td>
<td>General, in progress</td>
</tr>
<tr>
<td><strong>Optimizations:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Operation Minimization</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>• Memory Minimization</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td>• Space-Time Transformation</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>• Data Locality</td>
<td>Partial</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Parallel code generation</strong></td>
<td>Limited general</td>
<td>General, in progress</td>
</tr>
</tbody>
</table>
A High-Level Language for Tensor Contraction Expressions

range V = 3000;
range O = 100;
index a,b,c,d,e,f : V;
index i,j,k : O;
mlimit = 1000000000000;
function F1(V,V,V,O);
function F2(V,V,V,O);
procedure P(in T1[O,O,V,V], in T2[O,O,V,V], out X)=
begin
  X == sum[ sum[F1(a,b,f,k) * F2(c,e,b,k), {b,k}] * sum[T1[i,j,a,e] * T2[i,j,c,f], {i,j}], {a,e,c,f}];
end

\[ A3A = \frac{1}{2} X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} \]

\[ X_{ce,af} = t_{ij}^{ce} t_{ij}^{af} \quad Y_{ae,cf} = \langle ab \parallel ek \rangle \langle cb \parallel fk \rangle \]
TCE Components

- **Algebraic Transformations**
  - Minimize operation count

- **Memory Minimization**
  - Reduce intermediate storage

- **Space-Time Transformation**
  - Trade-offs btw storage and recomputation

- **Storage Management and Data Locality Optimization**
  - Optimize use of storage hierarchy

- **Data Distribution and Partitioning**
  - Optimize parallel layout

**Software Developer**

- Sequence of Matrix Products
- Element-wise Matrix Operations
- Element-wise Function Eval.

**Tensor Expressions**

- **Algebraic Transformations**
  - Minimize operation count

- **Memory Minimization**
  - Reduce intermediate storage

- **Space-Time Transformation**
  - Trade-offs btw storage and recomputation

- **Storage Management and Data Locality Optimization**
  - Optimize use of storage hierarchy

- **Data Distribution and Partitioning**
  - Optimize parallel layout

**System Memory Specification**

- No sol’n fits disk
  - Sol’n fits disk, not mem.
  - Sol’n fits mem.

**Space-Time Trade-Offs**

- No sol’n fits disk
- Sol’n fits mem.

**Storage and Data Locality Management**

- Sol’n fits mem.

**Data Distribution and Partitioning**

- Parallel Code
  - Fortran/C/…
  - OpenMP/MPI/Global Arrays

**Performance Model**
## Operation-Minimal and Memory-Minimal Forms

for a, e, c, f
  - for i, j
    - \( X_{aecf} += T_{i a e} + T_{j c f} \)
  - for c, e, b, k
    - \( T_{1c e b k} = f_{1}(c, e, b, k) \)
for a, f, b, k
  - \( T_{2afbk} = f_{2}(a, f, b, k) \)
for c, e, a, f
  - for b, k
    - \( Y_{ceaf} += T_{1 c e b k} + T_{afbk} \)
for c, e, a, f
  - \( E += X_{aecf} + Y_{ceaf} \)

### Array, Space, Time

<table>
<thead>
<tr>
<th>Array</th>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>( V^4 \rightarrow 1 )</td>
<td>( V^4 O^2 )</td>
</tr>
<tr>
<td>( T1 )</td>
<td>( V^3 O \rightarrow VO )</td>
<td>( C_{f1} V^3 O )</td>
</tr>
<tr>
<td>( T2 )</td>
<td>( V^3 O )</td>
<td>( C_{f2} V^3 O )</td>
</tr>
<tr>
<td>( Y )</td>
<td>( V^4 \rightarrow 1 )</td>
<td>( V^5 O )</td>
</tr>
<tr>
<td>( E )</td>
<td>1</td>
<td>( V^4 )</td>
</tr>
</tbody>
</table>

\[
A3A = \frac{1}{2} X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf}
\]

\[
X_{ce,af} = t_{ij} t_{ij}^a f \quad Y_{ae,cf} = \langle ab \rangle e k \langle cb \rangle f k
\]

for a, f, b, k
  - \( T_{2afbk} = f_{2}(a, f, b, k) \)
  - for c, e
    - \( T_{1 bk} = f_{1}(c, e, b, k) \)
  - for b, k
    - \( Y += T_{1 bk} + T_{afbk} \)
  - for a, f
    - \( X += T_{i a e} + T_{j c f} \)
for b, k
  - \( Y += T_{1bk} + T_{afbk} \)
  - for c, e
    - \( Y += T_{1 bk} + T_{afbk} \)
for a, f
  - \( X += T_{i a e} + T_{j c f} \)
for c, e
  - \( Y += T_{1 bk} + T_{afbk} \)

\[ E += X Y \]
Tiling provides a controlled compromise between minimal operations and minimal memory (full fusion)
Methods Implemented to Date using TCE

- CCD, CCSD, CCSDT, CCSDTQ
- iterative MBPT(2), MBPT(3), MBPT(4)
- EOM-CCSD, EOM-CCSDT, EOM-CCSDTQ
- CCSD-lambda/dipole, CCSDT-lambda/dipole, CCSDTQ-lambda/dipole
- LCCD, LCCSD
- QCISD
- CISD, CISDT, CISDTQ
- one-component relativistic of all of the above
- two- and four-component relativistic except for EOM-CC (in UTChem), CCSD(T), CCSD[T]
- integral-direct CCSD, avoiding $<ab||ci>$ and $<ab||cd>$ integrals
- Localized orbital/AO CCSD/MBPT[2] (currently sequential only)
- CCSD(T) using AO basis $<ab||cd>$
- CEPA, CEPA(T)
Parallel Scalability of Prototype TCE-Generated Code

**OH radical CCSDT/aug-cc-pVQZ**

- **Ideal Linear Speedup**
- **Number of Processors**
- **Relative speed**

- 0 50 100 150 200 250 300

- 0 1 2 3 4 5 6 7 8 9

- 16 42 74 128 256

*Courtesy of So Hirata, PNNL. Obtained on the HP Supercluster at PNNL’s Molecular Science Computing Facility*
range $V = 3000$;  
range $O = 100$; 

index $a,b,c,d,e,f : V$;  
index $i,j,k : O$;  

$\text{mlimit} = 1000000000000$; 

function $F1(V,V,V,O)$;  
function $F2(V,V,V,O)$;  


begin 
$X == \text{sum}[ \text{sum}[F1(a,b,f,k) \times F2(c,e,b,k), \{b,k\}]$  
* $\text{sum}[T1[i,j,a,e] \times T2[i,j,c,f], \{i,j\}]$,  
$\{a,e,c,f\}]$; 
end 

$A3A = \frac{1}{2} (X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf}$  
$+ X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf} + X_{ce,af} Y_{ae,cf})$ 

$X_{ce,af} = t_{ij}^{ce} t_{ij}^{af}$  
$Y_{ae,cf} = \langle ab \| ek \rangle \langle cb \| fk \rangle$
On the Drawing Board...

- More flexibility in sequencing and controlling optimizations
- Common sub-expression elimination
- Global factorization (across equations)
  - Complex problem
- Improving parallel code generation
  - Multi-level parallelism
    - Threads
    - Multiple loosely coupled tasks
- More sophisticated performance models
- Develop approximate algorithms for opt.
  - Address situations where exhaustive search too expensive
    - i.e. Deliver best result spending at most 3 min on code gen.
    - … or 60 min … or 3 days …
- Generalizations beyond electronic structure
TCE Summary

• Automatic generation of code from high-level algebraic expressions
  – Approach problem like a compiler
  – Use of “high-level language” allows automation of design decisions usually made by human software developer
  – Produce robust, reliable code

• Addresses productivity, complexity, and performance
  – Compiler-like optimizations key to full utility of code generation approaches

• Strong interdisciplinary collaboration between chemists and computer scientists
  – Formulation & understanding of problem from chemists
  – Solutions from computer scientists (w/ significant help from chemists)

• Helping to bring CC theory back to nuclear physics
Looking to the Future
(Summary)

• High-performance computer architectures will not get simpler
  – New programming models can help make efficient programming of them easier by raising the level of abstraction
  – E.g. Global Arrays, Tensor Contraction Engine, etc.

• Increasing push for multi-scale, multi-physics simulations, interoperability, collaboration around software
  – Component architectures facilitate all of these
  – E.g. Common Component Architecture

• Strong collaborations between domain scientists and computer scientists produce tools/environments for users